

UNITED STATES PROVISIONAL PATENT APPLICATION

FOR

**METHODS AND SYSTEMS FOR ACCESSING, ORGANIZING,  
PRESENTING AND VIEWING DATA**

INVENTORS:

Ali Kutay of Palo Alto, California

Cihan Akin of Redwood City, California

Eliahu Albek of San Francisco, California

Erhan C. Akin of Foster City, California

Hakan Akin of San Mateo, California

John Gilbert of Belmont, California

Stephen Wilkes of Santa Clara, CA 95051

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN  
12400 WILSHIRE BOULEVARD  
SEVENTH FLOOR  
LOS ANGELES, CALIFORNIA 90025  
(408) 720-8598

Attorney's Docket No. 003866.P005Z

"Express Mail" mailing label number: EL471466729US

Date of Deposit: June 5, 2000

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Assistant Commissioner for Patents, Washington, D. C. 20231

Lindy Varetta

(Typed or printed name of person mailing paper or fee)

Lindy Varetta

(Signature of person mailing paper or fee)

6-5-00

(Date signed)



White Paper

# Next Generation eBusiness Development

## The AltoWeb Platform

*A primer on how the AltoWeb Platform can help your enterprise to easily create industry leading eBusiness applications, while solving many of today's technology and business challenges.*

# Introduction

The need to create effective, engaging eBusiness applications is placing incredible demands upon the resources of the enterprise. Success, in this competitive market, belongs to those who are able to implement eBusiness applications *before* their competition.

This new breed of complex eBusiness applications requires immediate information availability from web servers, file servers, databases, and legacy systems throughout the enterprise.

## Legacy Systems

Information today is stored inside and outside of corporate walls -- on systems belonging to the enterprise, but also to partners, suppliers, and customers. ERP, SFA, CRM, inventory, and other disparate systems need to be accessed, and the heterogeneous information that results from these queries must be processed and delivered. The rendered solution may also need to be formatted for multiple outputs, such as web browsers, wireless communications, or other electronic devices.

This entire process is usually facilitated by a vast amount of non-reusable custom coding. Even when this work is reusable, updating, maintaining and troubleshooting this code is often a full time job.

## Partial Solutions

Application servers, middleware companies, EAI vendors, other service and software providers all have partial solutions to these challenges, but none provide a comprehensive platform for eBusiness development that satisfies all of them.

## A Complete Solution

The AltoWeb Platform provides a comprehensive, sophisticated system that addresses the needs of creating complex eBusiness applications.

Many business and technological challenges associated with the development and deployment of today's eBusiness applications are solved by AltoWeb's technological innovations (patents applied).

For example, AltoWeb's advances in server technology have resulted in the creation of "uniform data objects." These unique data objects enable real-time data access, management, and transformation capabilities far in excess of any solution offered in the marketplace today.

This means that the critical time-to-market for eBusiness development can be dramatically reduced.

## Visual Development

AltoWeb's visual development environment, AltoStudio, features "drag and drop" functionality, and dynamic Java and SQL code generators. Use of the studio enables multiple format data rendering such as XML, HTML, WML, JSP, ASP, and other custom output formats.

AltoWeb simplifies the data rendering process by separating the presentation logic from the business logic of an application.

AltoWeb has also created advances in navigation technology, which allow organizations to present information to an end user in an entirely new way. The user has increased control of the navigation process, and their screen landscape is maximized. This means that large amounts of information are available for easy viewing, searching, and transacting.

Enterprises that choose to deploy AltoWeb-based solutions will experience substantial improvements in flexibility, power, and accelerated development of cutting-edge, real-time eBusiness applications.

## Today's eBusiness Limitations

Application servers have added significant power to the creation of eBusiness applications by facilitating and improving database accessibility. Their strength also resides in their ability to provide shared services such as connection pooling, business rule storage and execution, resource management, and security. On the other hand, application servers still require the development of custom code to access databases or implement business rules that extract the data. In fact, depending on the application server, the amount of custom code development can be prohibitive.

In addition to writing code to move real-time data from the back-end to the front-end, a developer must also write custom code to apply additional business logic, such as adding all the rows in a column. Current technologies require developers to write

custom code each time they combine result sets coming from different data sources. For example, the effort to dynamically combine a query result from a database with a search result from a file server can easily take several weeks. There is simply no existing technology that treats objects that refer to different data sources uniformly -- you cannot relate a database object to a web object without a major development effort.

## AltoWeb's Advantage

- **The AltoWeb eBusiness Platform reduces application deployment time .**

This is possible because AltoWeb enables developers to create end-to-end web applications without writing a single line of code.

- **The AltoWeb eBusiness Platform can dynamically access, aggregate, structure, relate, and combine real-time data from multiple data sources and provide the resulting information in multiple output formats.**

AltoWeb has achieved this functionality by successfully separating business logic from presentation logic. Data objects are defined in AltoStudio and return a generic result set. Presentation logic then takes the result set, formats it according to the presentation templates defined within AltoStudio, and forwards it to the clients.

- **AltoWeb developed the concept of an abstract data source. This means that each disparate data source can be represented by a similar data source object.**



By applying this abstraction to back-end data sources such as databases, web servers, file servers, and XML servers, AltoWeb is able to treat all data sources uniformly, *regardless of the platform or type of data source.*

- AltoWeb's technology is able to map all data to uniform data objects. It has the ability to aggregate, combine, and relate data objects regardless of their underlying type.

For example, you can create a relationship between a database object and a file server object.

- AltoWeb delivers up-to-date, real-time content.

AltoWeb's technology uses data objects that reference actual data in the back-end, rather than replicated data from an original source. This means that the information presented to end users at run-time is completely dynamic. As a result, a change to the data at the back-end does not require any additional changes for presenting the information. Updates to back-end data are reflected at the front-end.

- AltoWeb data objects have the benefit of being reusable.

For example, data objects 1, 2, and 3 can represent one way of presenting information while the same data objects 1 and 2 can be reused along with object 4 to present the information in another way. If there is a change to data that is referred by object 1, the change will be reflected in both presentations without requiring any additional changes.

- AltoWeb supports multiple output formats including XML, HTML, WML, JSP, and ASP.

AltoWeb's support of XML in particular, allows custom output formats to be created by defining a new stylesheet. For example, it is possible to publish a result set in XML and combine it with an XSL to create HTML output. At the same time, the XML result can be combined with another XSL to create WML output.

- AltoWeb uses open standards such as servlets and EJBs.

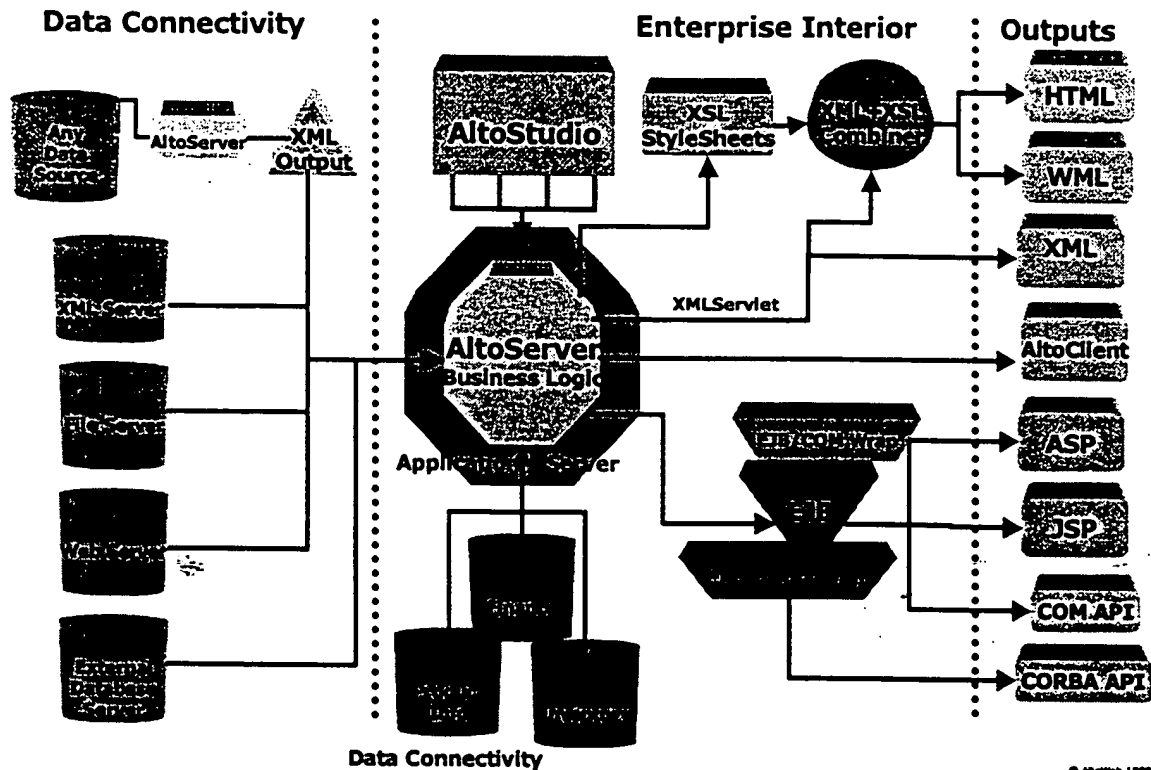
This allows the AltoWeb architecture to be highly scalable, both horizontally and vertically.

- AltoWeb features a component-based architecture (see next section for layout). This architecture enables a solution to be highly extensible.

If an organization needs to add support for a custom data source, it can develop and plug in a data connector that supports the custom data source. Alternately, if an organization needs to support a custom output format (e.g., publishing the results as a Word document), it can develop and plug-in a servlet that supports the required output format.

- AltoWeb technology uses standard Internet protocols. This enables access to data sources that are internal as well as external to an organization.

For example, if a supplier publishes its inventory on an XML server, AltoWeb technology allows the buyer to define an access path to the external XML server. Once the external data source is defined, developers can create data objects and apply business rules to the data objects, similar to data objects that refer to internal data sources.



© AltoWeb 1999

## AltoWeb's eBusiness Platform

The AltoWeb eBusiness Platform provides a rapid application development solution that enables customers to access, aggregate, structure, and relate data from disparate data sources.

The AltoWeb eBusiness Platform works in conjunction with existing application servers and leverages such shared services as connectivity, resource management, security, and transactional capabilities.

The AltoWeb eBusiness Platform consists of three main components: **AltoServer**, **AltoStudio**, and **AltoClient**.

**AltoServer** is a cornerstone of AltoWeb's technology. It provides real-time access to back-end data sources. Supported data sources include database servers such as Oracle, MS SQL, Informix, Sybase, DB2, as well as file servers, web servers, and XML servers. AltoServer also provides file and naming services to both **AltoClient** and **AltoStudio**. AltoServer supports standard output formats such as XML, HTML, WML, JSP, ASP, as well as additional AltoWeb defined data formats.

**AltoStudio** is a visual development environment that enables system administrators, developers, and web site designers to quickly create and deploy web applications. It provides a user-friendly drag and drop interface which

enables users to create web experiences within a very short time.

AltoStudio allows developers to access multiple data sources, create objects that refer to data sources, define the business rules for selecting subsets of information, and create web applications, all without writing a single line of code.

AltoStudio generates Java code and SQL statements dynamically, and provides an interface, which enables developers to modify the generated code.

AltoStudio allows users to assign visual properties to objects such as background color, font size, font color, etc. In addition, AltoStudio enables developers to create reusable presentation templates and map data objects to the templates. This means that developers can define how the information will be presented to end users, and then reuse that template for another application. Different data objects can be mapped to the same template to provide different presentations of information.

Once a user has created objects and defined the presentation of the result sets, the information is saved on AltoServer.

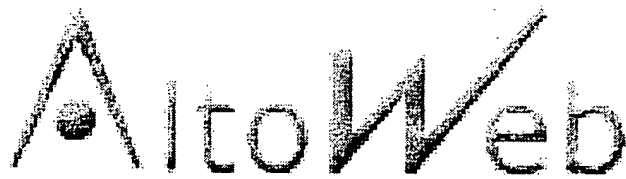
AltoClient is a next generation user interface that has the unique ability to map result sets onto three-dimensional shapes. AltoClient communicates with AltoServer to invoke queries. These queries provide result sets to AltoClient. AltoClient can display large amounts of data in a limited area. This enables end users to interact and navigate with the information in a highly efficient way.

AltoWeb currently supports several different renditions of AltoClient, including three-dimensional spheres and cubes. The resulting web presentation is highly differentiated from today's HTML

pages, and the navigation capabilities will redefine how end users browse, search, and move through enormous amounts of information on the web.

## Conclusion

AltoWeb has developed a platform and run-time environment that features real-time data access, management, transformation and presentation capabilities.



*AltoStudio*

**User's Guide**

AltoWeb, Inc.  
1731 Embarcadero Road  
Palo Alto, California 94303  
650.855.8880  
[www.altoweb.com](http://www.altoweb.com)

---

---

# *Contents*

---

## **CHAPTER 1**

### *Overview* **9**

#### **Introduction 10**

#### **Welcome to AltoWeb 10**

##### *Implementation 10*

#### **AltoClient 11**

##### *Three Layers Of Abstraction 12*

#### **AltoStudio 17**

#### **AltoServer 17**

#### **AltoStudio Components 18**

#### **Sections 19**

##### *Data Modeler 19*

##### *Layout Designer 19*

##### *DataView Designer 19*

#### **System Requirements 20**

#### **Definitions 20**

##### *AltoWeb Applet 20*

##### *MetaObject 21*

##### *MetaObject Connections 21*

**CHAPTER 2**

***AltoStudio Window 23***

**Menu Bar 24**

*File 25*

*Edit 26*

*View 27*

*Insert 28*

*Tools 29*

*Window 30*

*Help 30*

**Quick Access Icons 31**

**Quick Access Icon Dialog Boxes 33**

**Access Flexibility 35**

**CHAPTER 3**

***Connecting to a Server 39***

**Flow Diagram 40**

**Initializing AltoStudio 41**

**Starting AltoStudio 42**

*runStudio.bat Window 42*

**Connecting to a Server 43**

**CHAPTER 4**

***Creating a Project 47***

**Creating a Project 47**

*New Project 47*

*Open Project 51*

**CHAPTER 5**

***Connecting to a Data Source 55***

**Connecting to a Data Source 56**

**New Project Data Source 56**

*DataSource Wizard 59*

*Test 62*

**WebLogic Properties File 63**

**Data Source Properties 65**

*Modifying a DataSource 66*

**Inserting a New DataSource 66**

**DataSource Wizard 67**

*Drop-down Alternatives 68*

**CHAPTER 6**

***MetaObjects 71***

Background **72**

MetaDefinitions **72**

*MetaObject 72*

*MetaObject Connection 73*

Creating MetaObjects **73**

Insert > New MetaObject **74**

MetaObject Wizard **75**

*Name and Type tab 76*

*Choose Sources tab 78*

*Define Joins tab 79*

*Define Fields tab 83*

*Visual Tag 85*

*Define Logic Tab 86*

*Define Queries Tab 90*

Quick Access Add Visual Tabs Icon **93**

MetaObject Wizard Icon **97**

**CHAPTER 7**

***Connecting MetaObjects 101***

Accessing Connections **102**

Relationship Diagram **102**

Connection Wizard **105**

*Set Connection Properties tab 106*

*Configure Query tab 107*

**CHAPTER 8**

***Templates and Screens 109***

To View or Modify an Existing Template **110**

Creating a Template **111**

Insert > New Template **111**

Quick Access Template Editors Icon **112**

*Template Editor — NewFacetPopup 113*

*Code Editor 118*

*Template Editor — New Context 119*

*Template Editor — NewStructure 122*

Insert > New Screen **125**

Quick Access Screen Editors Icon **125**

*Context Screen Properties 125*





---

This chapter includes the following topics:

- "Introduction" on page 10
- "Welcome to AltoWeb" on page 10
- "AltoStudio Components" on page 18
- "Sections" on page 19
- "System Requirements" on page 20
- "Definitions" on page 20

---

## *Introduction*

At the highest level, the objectives for AltoStudio are:

- To access, relate, and structure information from diverse databases, web files, and multimedia servers
- To apply visual properties to this data
- To provide a highly customized user interface which enables end users to navigate and interact with this data in a straightforward and efficient manner

---

## *Welcome to AltoWeb*

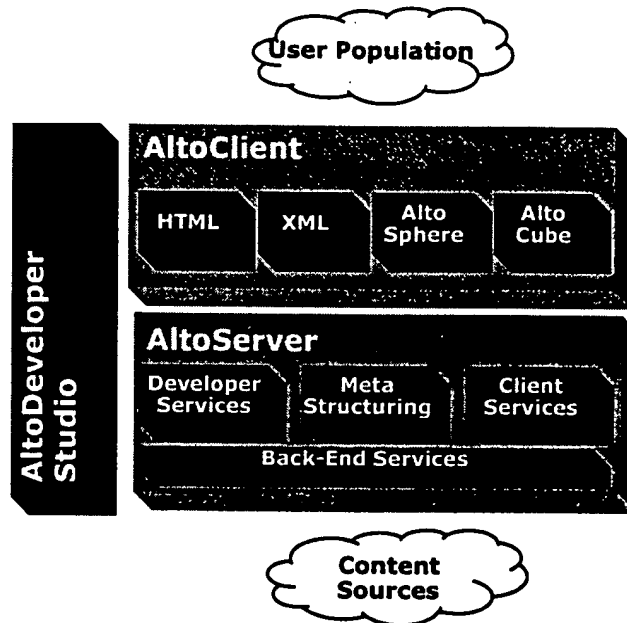
AltoWeb has developed a three-tiered technology which, at the user level, is reflected by an innovative user interface termed the AltoClient. The technology that underlies the AltoClient and makes it possible is two-fold: AltoServer and AltoStudio.

## **Implementation**

The implementation of the objectives of the AltoStudio must begin with the goal of determining just what, exactly, the composition of the user interface should be.

The issue is to determine what information, textual as well as graphic, should be included in that interface and how it should be presented. These objectives are created within the AltoStudio and ultimately reflected by the AltoClient.

The overall AltoWeb technology and its interrelationships with the AltoStudio, AltoServer and AltoClient can be graphically shown in the following diagram.



---

## AltoClient

The bottom line for the user is the AltoClient: AltoWeb's graphic interface. AltoClient, as well as the developer and server technology that supports it — is useful — as a visualization and extraction tool, it lets you see both the conventional overall design of a web site yet, at the same time, allows you to navigate, exactly and quickly, to the specific areas you are interested in.

That is, instead of sifting through an interminable list of bulleted subsections and drop-down category bars, you can use a stylish, intuitive overview in the form of a three-dimensional sphere, cube, or HTML/XML format, to determine the specific information that you need.

AltoClient makes web sites easier to navigate by clearly laying out their entire structure. While simple organization charts, decision trees and box-and-line diagrams work for small sites, at larger sites they can often be tedious.

For example, you can spend a lot of time following link after link from a home page, navigating through numerous charts and diagrams, to ultimately find what you want.

Or you can use AltoClient.

### Three Layers Of Abstraction

The AltoClient UI consists of three panes. Located from left to right they are: Structure, Context, and Content.

1. Structure — The hierarchical relationship of rendered facets to other facets is displayed in the left pane.

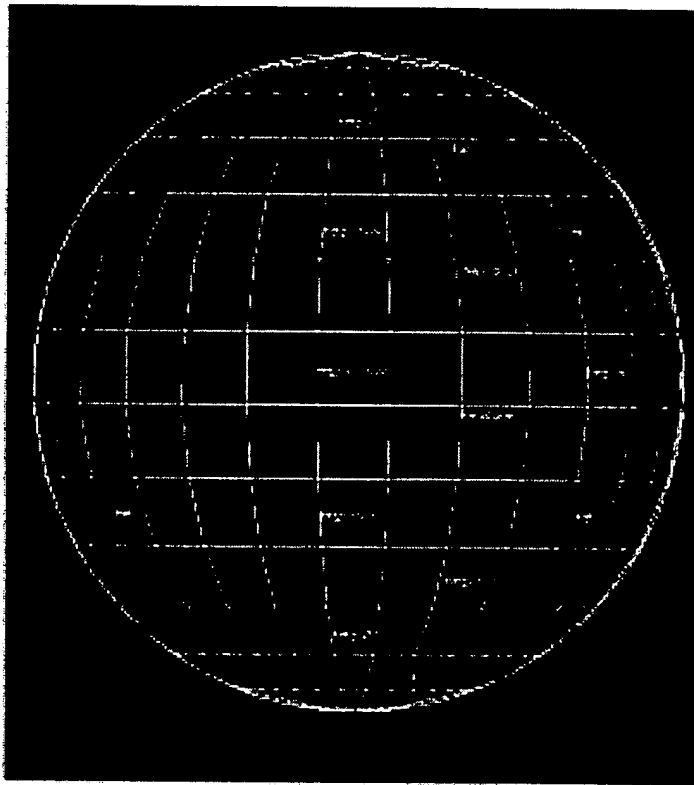
Facet rendering is the visual display representing a structured link to additional hierarchical structures or terminal nodes of content.



Structure is similar in concept to a "table of contents" or master tree. It represents the manner in which a web site has defined its contents for browsing. In this case, structure is dynamically driven by MetaData applications previously determined in the AltoServer.

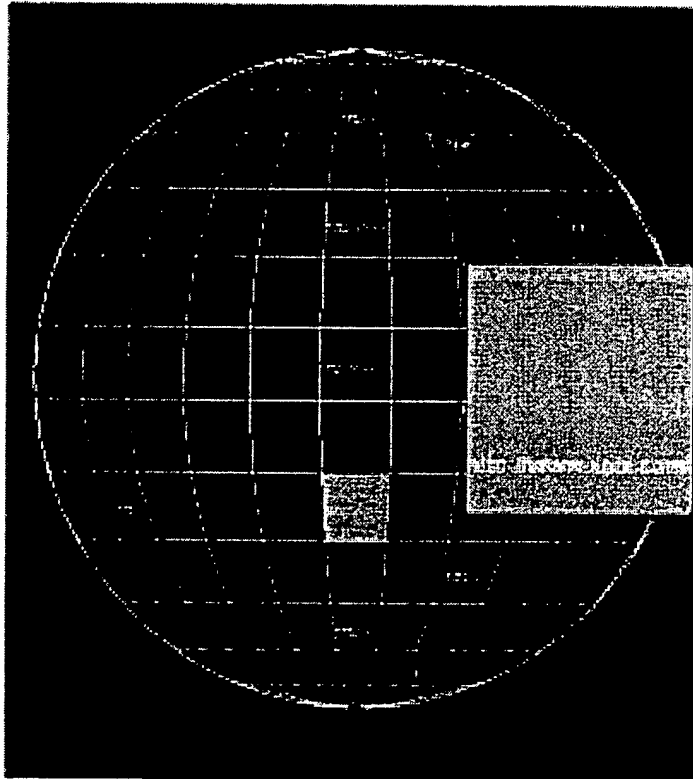
In short, MetaClusters are viewed in the master tree structure paned.

2. Context — This is the center pane in the interface. It contains rendered capabilities including the oriented displays of facets and related linked contents.

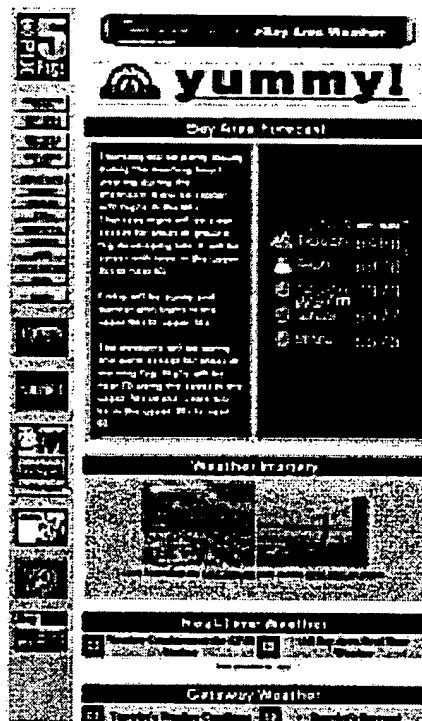


An AltoSphere, shown above, is a spherical representation of facets. There are several ways to render this context besides an AltoSphere: an AltoCube (a cubic representation of facets), or HTML/XML (a standard HTML/XML representation of facets).

- **Popup** - Placing your cursor on any facet with a visual representation in the sphere will produce a pop-up dialog box. This dialog box contains user-determined information about the contents of that facet or links to other information.



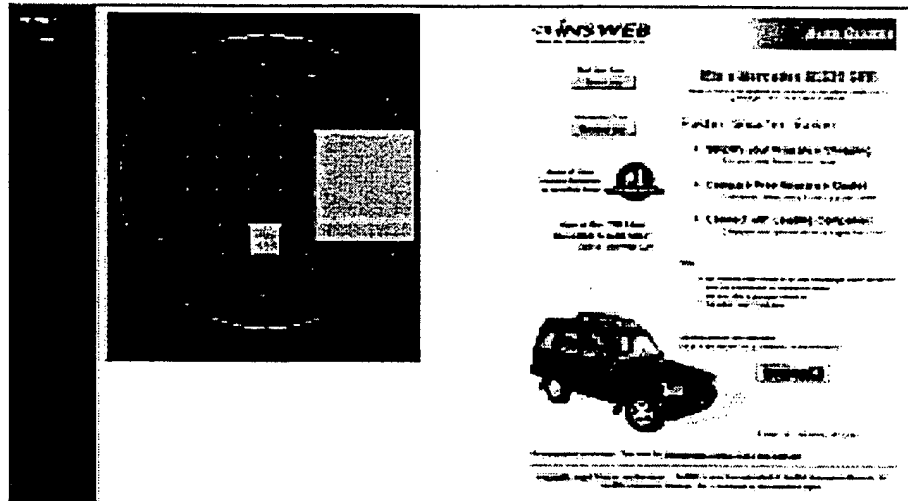
3. Content - The right pane, Content, represents all source information which has been refined by the structure and context panes. In this example, the facet has opened a link to a local weather station.



In summary, AltoClient is the "front-end" user interface that incorporates a family of dimension-capable browser plug-ins or Java applets that maps query results onto 3-D shapes or HTML/XML documents.

These documents, in turn, present web-site content in a unique tri-pane window that includes a user-determined spherical or cubic view for ease-of-navigation.

Although a specific interface will be user-determined to be particular to your environment, a typical AltoClient interface might resemble the following:



Although any public domain URL could have been used, in this example we selected the URL *www.insweb.com*. Just as importantly, many URL's could have been selected, with each occupying a separate facet on the sphere.

Specifically, the left pane (structure) shows you the contents of your selection. It represents a table of contents that is visually implemented in the center and right panes.

The center pane (context), a sphere in this case, consists of the information or URL sites you have transferred to individual facets on the sphere.

You will note that the cursor has highlighted the URL *www.insweb.com*; and that a subsequent pop-up displays user-determined information within that pop-up.

The right pane (content), in this instance, shows the real-time contents to the URL site.



---

## *AltoStudio*

AltoStudio enables system administrators and Web site designers to:

- Create data models from multiple and diverse information sources
- To specify the operations and transformations that are to be applied to the data
- Customize the presentation of the data model for the end user

AltoStudio allows you to map and customize the meta-data schema to an existing application and data source with minimal effort. This can be defined as software that allows the rapid development environment of content aggregation and configuration of the AltoClient.

AltoStudio can also be defined as a dynamic developmental environment from which you define data sources and, subsequently, forward those sources to web site developers for web page content.

In short, AltoStudio can be either a developer or a nondeveloper solution that is used to create the content and look of AltoClient.

---

## *AltoServer*

AltoServer is a cornerstone of the AltoWeb solution. It provides services such as connecting data sources as well as loading and saving existing projects both for AltoStudio and AltoClient.

AltoServer provides the underlying technology that supports the connectivity to data sources. It both saves and loads projects for the AltoStudio and AltoClient.

AltoServer marshals data from distributed, heterogeneous data sources and forwards that information into a medium suitable for

Web publication. It also leverages application server applications such as security database connectivity and naming services.

Examples of data sources that can be used by AltoServer include:

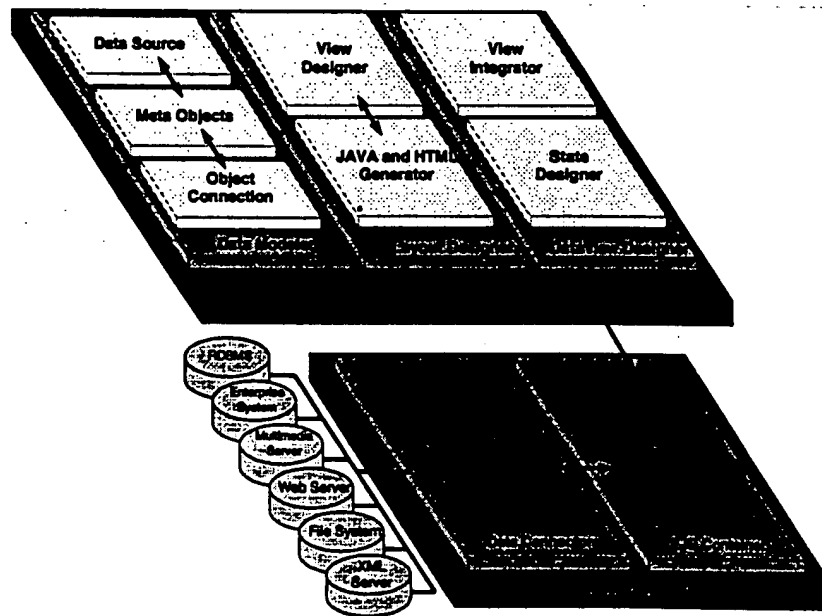
- RDBMS
- Video and other data streams
- Enterprise (ERP) systems

---

## *AltoStudio Components*

---

The overall relationship between the AltoStudio and the AltoServer can be shown in the following graphic.



---

## *Sections*

AltoStudio consists of three major sections:

### **Data Modeler**

The Data Modeler is used to:

1. Create connections to data sources by allowing you to specify:
  - Data sources
  - Paths to data sources, and
  - Authentication parameters for data sources
2. Define MetaObjects that contain references to data sources.
3. Create connections between MetaObjects which, in turn, define relationships among them.

### **Layout Designer**

The Layout Designer incorporates editors and wizards that allow you to create templates including:

- An overall view of the result set or data items
- The visual properties of each data item

### **DataView Designer**

The DataView Designer includes editors and wizards that allow you to:

1. Create connections between MetaObjects.
2. Establish visual properties of the site.
3. Define the relationships among MetaObjects into a site map.

---

## *System Requirements*

System requirements will vary from platform to platform. In general, however, AltoStudio requires:

- An Intel Pentium processor
- 2 MB hard drive disk space
- Windows 95, 98, NT 4.0
- JDK 1.1.7
- 64 Mb of RAM
- Microsoft Internet Explorer (4.0.1+)
- Netscape Navigator (4.0+)

---

## *Definitions*

Once AltoStudio has defined your solution, AltoClient allows you to navigate and interact with AltoWeb MetaObjects. The following definitions are specific to this process.

### **AltoWeb Applet**

An AltoWeb applet is a program written in the Java Programming language that can be included in an HTML page, much in the same way an image is included. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser.

Specifically, AltoClient runs as an applet inside a browser. And, as a client applet, it remains dependent on the AltoServer.

## MetaObject

A MetaObject is a descriptor to a specific connection containing selection criteria. It may also contain fields from a single data source including information in a table or hyperlinks in an HTML document.

For example, a MetaObject might consist of a selection of all automobiles where the *location = germany*. Or all hyperlinks that start with *www.gullwing.com*.

MetaObjects can also have additional constraints including, for example, all *automobiles = gran touring = german = gullwing*.

## MetaObject Connections

The connections or relationships between MetaObjects is known as a MetaObject connection.

A MetaObject connection query describes how two, or more, MetaObjects are related to each other.

The following Chapter: "AltoStudio Window" on page 23, begins the process of developing the AltoWeb solution.

1968	1967	1966	1965	1964	1963	1962	1961	1960
1959	1958	1957	1956	1955	1954	1953	1952	1951
1950	1949	1948	1947	1946	1945	1944	1943	1942
1941	1940	1939	1938	1937	1936	1935	1934	1933
1932	1931	1930	1929	1928	1927	1926	1925	1924
1923	1922	1921	1920	1919	1918	1917	1916	1915
1914	1913	1912	1911	1910	1909	1908	1907	1906
1905	1904	1903	1902	1901	1900	1899	1898	1897
1896	1895	1894	1893	1892	1891	1890	1889	1888
1887	1886	1885	1884	1883	1882	1881	1880	1879
1878	1877	1876	1875	1874	1873	1872	1871	1870
1869	1868	1867	1866	1865	1864	1863	1862	1861
1860	1859	1858	1857	1856	1855	1854	1853	1852
1851	1850	1849	1848	1847	1846	1845	1844	1843
1842	1841	1840	1839	1838	1837	1836	1835	1834
1833	1832	1831	1830	1829	1828	1827	1826	1825
1824	1823	1822	1821	1820	1819	1818	1817	1816
1815	1814	1813	1812	1811	1810	1809	1808	1807
1806	1805	1804	1803	1802	1801	1800	1799	1798
1797	1796	1795	1794	1793	1792	1791	1790	1789
1788	1787	1786	1785	1784	1783	1782	1781	1780
1779	1778	1777	1776	1775	1774	1773	1772	1771
1770	1769	1768	1767	1766	1765	1764	1763	1762
1761	1760	1759	1758	1757	1756	1755	1754	1753
1752	1751	1750	1749	1748	1747	1746	1745	1744
1743	1742	1741	1740	1739	1738	1737	1736	1735
1734	1733	1732	1731	1730	1729	1728	1727	1726
1725	1724	1723	1722	1721	1720	1719	1718	1717
1716	1715	1714	1713	1712	1711	1710	1709	1708
1707	1706	1705	1704	1703	1702	1701	1700	1699
1698	1697	1696	1695	1694	1693	1692	1691	1690
1689	1688	1687	1686	1685	1684	1683	1682	1681
1680	1679	1678	1677	1676	1675	1674	1673	1672
1671	1670	1669	1668	1667	1666	1665	1664	1663
1662	1661	1660	1659	1658	1657	1656	1655	1654
1653	1652	1651	1650	1649	1648	1647	1646	1645
1644	1643	1642	1641	1640	1639	1638	1637	1636
1635	1634	1633	1632	1631	1630	1629	1628	1627
1626	1625	1624	1623	1622	1621	1620	1619	1618
1617	1616	1615	1614	1613	1612	1611	1610	1609
1608	1607	1606	1605	1604	1603	1602	1601	1600
1599	1598	1597	1596	1595	1594	1593	1592	1591
1590	1589	1588	1587	1586	1585	1584	1583	1582
1581	1580	1579	1578	1577	1576	1575	1574	1573
1572	1571	1570	1569	1568	1567	1566	1565	1564
1563	1562	1561	1560	1559	1558	1557	1556	1555
1554	1553	1552	1551	1550	1549	1548	1547	1546
1545	1544	1543	1542	1541				

---

AltoStudio incorporates a Multiple Document Interface (MDI). MDI enables you to reposition multiple panes to suit your individual viewing requirements. You can also resize panes in the same window.

It will be helpful if you become familiar with the menu bar and Quick Access tool bar icons in the AltoStudio window before you begin.

This chapter includes the following topics:

- "Menu Bar" on page 24
- "Quick Access Icons" on page 31
- "Quick Access Icon Dialog Boxes" on page 33
- "Access Flexibility" on page 35

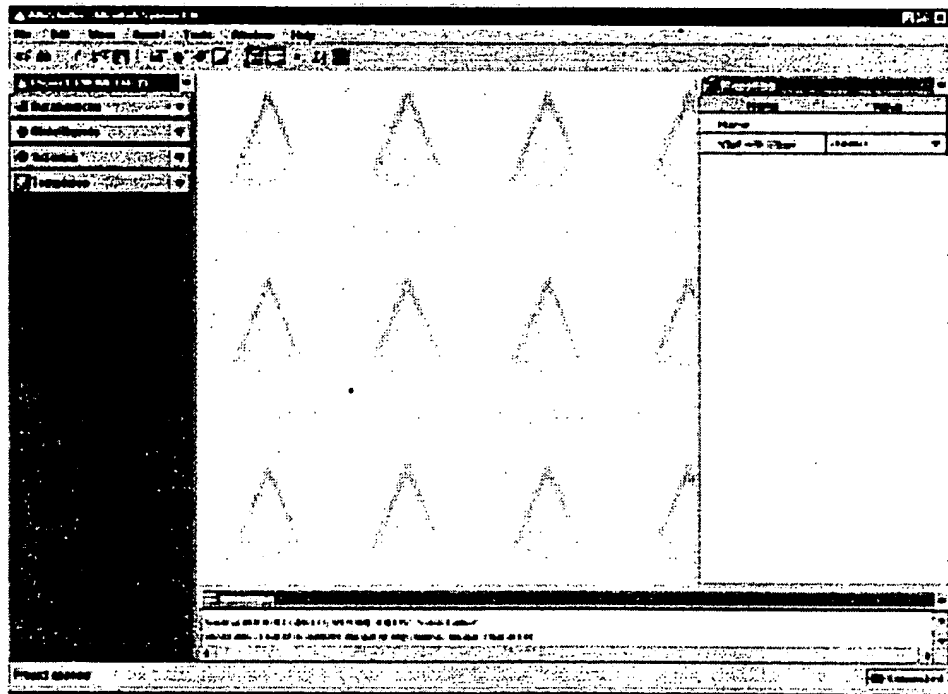
## *Menu Bar*

The AltoStudio Window has a menu bar with seven selections — each of which supports drop-down fields.

Directly below the menu bar is a Quick Access tool bar with fourteen icons (See “Quick Access Icons” on page 31.). Most of these icons duplicate, or enhance, the functionality of the selections found in the menu bar.

This section will address the information found in the menu bar.

A typical AltoStudio Window is shown below.



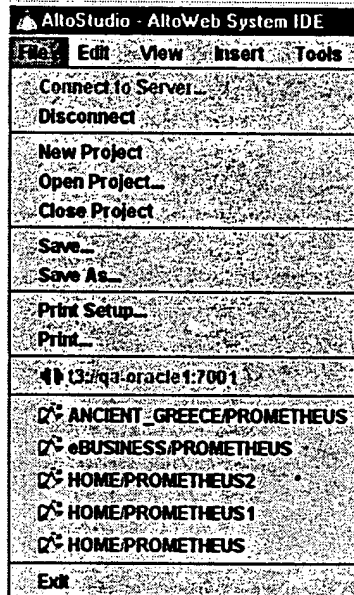


The AltoStudio window menu bar is shown below.



## File

The first entry, reading from left to right in the menu bar, is **File**.



**File** supports ten drop-down project alternatives:

- **Connect to Server** — Single clicking **Connect to Server** will open a dialog box from which you can enter the address of the server you want to use (normally it will be the location of the AltoServer). Chapter 3, "Connecting to a Server" on page 39, presents this process.
- **Disconnect** — Clicking this field allows you to disconnect from any server presently active.

- **New Project** — Accessing this field allows you to create a new project. Chapter 4 "Creating a Project" on page 47, discusses this process.
- **Open Project** — Clicking **Open Project** allows you to access any existing project previously created and saved.
- **Save** — Clicking **Save** will save any project you have open *and* leave the project open.
- **Save As** — Clicking **Save As** will save any project you have open but with a different name. All changes will be saved when the project is opened under the new name.
- **Print Setup** — This field allows you to specify print options such as how many copies to print, print crop marks, registration marks, and more. Available options vary slightly with the platform and the printer you are using.
- **Print** — To print a document, choose **File > Print**. Then specify the page range you want to print. Select the print options and click **Print**.
- **Exit** — Clicking **Exit** will terminate the session. All *unsaved* information will be lost and changes will not be retained.

## Edit

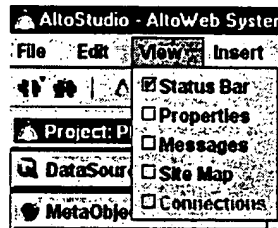
The second entry in the menu bar is **Edit**:



**Edit** supports industry-standard menu commands such as Cut, Copy, Paste, Delete, and Select All.

## View

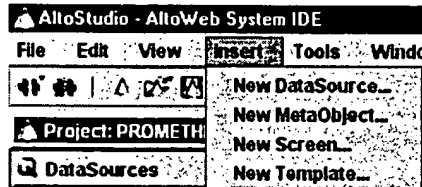
The third entry in the menu bar is **View**. There are five drop-down alternatives with a check box to the left of each.



- **Status Bar** — This is an information dialog banner normally shown at the bottom of the window. It shows whether the AltoServer is connected, or disconnected, to the server. It will also present connectivity information.
- **Properties** — This dialog box describes the existing identity and physical characteristics of the data that you are using. This can include the name of the project, the colour, the text font and more.
- **Messages** — All incoming messages will be shown this dialog box.
- **Site Map** — This window shows how the project is mapped to other projects.
- **Connections** — This window shows you how MetaObjects are connected together to form a site map.

## Insert

The forth entry in the menu bar is **Insert**.

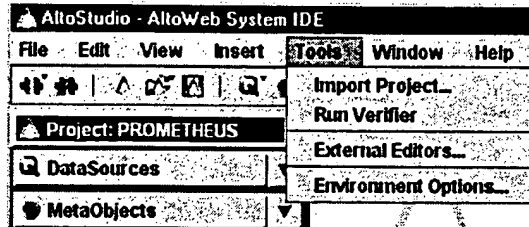


**Insert** supports four drop-down project alternatives:

- **New DataSource** — Clicking this field (or highlighting the field and pressing **Enter/Return**) will bring up the **Data Source Wizard** window (See "DataSource Wizard" on page 59.). The **Data Source Wizard** window allows you to access all data sources that are available through your server.
- **New MetaObject** — Clicking **New MetaObject** will bring up the **MetaObject Wizard** window (See "MetaObject Wizard" on page 75.). The **MetaObject Wizard** window supports six tabs. Each tab is used to determine a specific aspect of a MetaObject.
- **New Screen** — Clicking this field opens the Screen Editor. The Screen Editor is used to determine the layout of the cube or sphere.
- **New Template** — This function will open the Template Editors. The Template Editor consists of Structure, Context, and Facets. Structure is used to determine colors. Context represents the place holder for various properties such as colours and fonts for the sphere. Facets is used to design the facets within the bands and pop-ups.

## Tools

The fifth entry in the menu bar is **Tools**.



**Tools** supports four drop-down project alternatives:

- **Import Project**

This feature allows you to import a new project with a unique identification. This new project will, in turn, append to an existing project.

- **Run Verifier**

Run Verifier will ensure the integrity of your project by notifying you if there are obvious logical errors or omissions in the construction of your project.

- **External Editors**

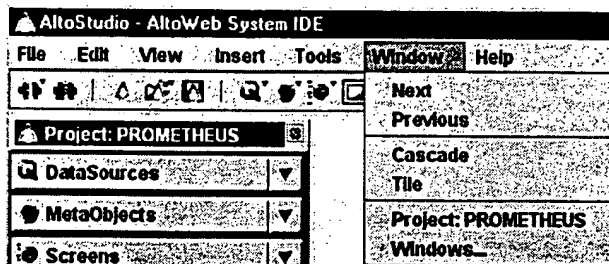
This will allow you to specify external editors for your internal Java code. Editors will consist of colour enhancements or graphics.

- **Environment Options**

This feature will clear your server and project lists.

## Window

The sixth entry in the menu bar is **Window**.

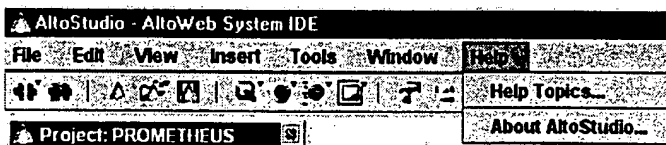


**Window** supports five drop-down project alternatives:

- **Next** — Shows the subsequent project window in the sequence.
- **Previous** — Shows the previous project window in your sequence.
- **Cascade** — Use this command to arrange multiple opened windows into an overlapped fashion.
- **Tile** — Use this command to arrange multiple opened windows into a non-overlapped fashion.
- **Windows** — Create a new window that views the same images.

## Help

The last entry in the menu bar is **Help**.



While the implementation of On-line help will be fully developed in a subsequent release, it is presently possible to view the Version number of AltoStudio by clicking the field About AltoStudio.

## Quick Access Icons

Fourteen Quick Access tool bar icons are located directly below the AltoStudio window menu bar.



The functionality and identity of these icons is summarized in Table 1. Note that the MetaObject Wizard icon and the DataSource Wizard icon are not active until after a project is created.

TABLE 1. Quick Access Icons






ICON	DIALOG	EXPLANATION
	Connect to Server	Same functionality as: File > Connect to Server
	Disconnect	Same functionality as: File > Disconnect to Server
	New Project	Same functionality as: File > New Project
	Open Project	Same functionality as: File > Open Project
	Save Project	Same functionality as: File > Save Project

TABLE 1. Quick Access Icons (Continued)










ICON	DIALOG	EXPLANATION
	DataSource Wizard	Supports four drop-down icons: New SQL/JDBC DataSource New File System DataSource New Web Server DataSource XML DataSource
	MetaObject Wizard	Supports four drop-down icons: DataSet MetaObject Compound MetaObject Linked MetaObject Custom MetaObject
	Screen Editors	Same functionality as: Insert > New Screen. Supports: Context Editor Context Screen Properties
	Template Editors	Same functionality as: Insert > New Template Supports three drop-down icons: Facet Template Editor Context Template Editor Structure Template Editor



TABLE 1. Quick Access Icons (Continued)

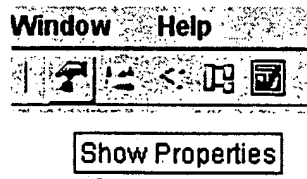
ICON	DIALOG	EXPLANATION
	Show Properties	Same functionality as: View > Properties
	Show Messages	Same functionality as: View > Messages
	Show Site Map	Same functionality as: View > Site Map
	Show Connections	Same functionality as: View > Site Connections
	Add Visual Tabs	Used to position information on the facets

## Quick Access Icon Dialog Boxes

Selecting and clicking a Quick Access icon will generally produce a dialog box which, in turn, contains additional fields.

For example: Select and click the **Show Properties** Quick Access icon.

**NOTE:** This is basically the same as the menu bar alternative: **AltoStudio > View > Properties.**



This will open a Properties icon dialog box.

Properties	
Name	Value
Name	Web Server
TypeName	Web Server
➤ Source Specific	
URL	http://www.yahoo.com/

The information in the Properties dialog box fields can be changed in accordance with the project that is open. Values and fields that are not grayed out can be edited.

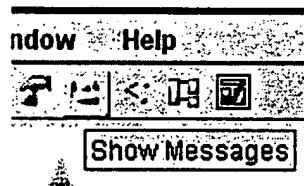
Since, for example, the property value TypeName "Web Server" is grayed-out then it is not possible to edit that value.

On the other hand, the value in a Name field can be changed from Web Server to any more appropriate name you want. To change a value name, highlight the field, click backspace, type your change and press **Return** or **Enter**.

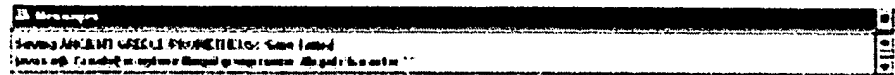
Another example is **Show Messages**.

Select and click the **Show Messages** Quick Access icon.

**NOTE:** This is basically the same as the menu bar alternative:  
**AltoStudio > View > Messages**.



This will open the **Messages** dialog box.



In this example the message tells you that the project ANCIENT GREECE/PROMETHEUS failed to save. The exception is based upon the fact that it contains a space in the title between ANCIENT and GREECE.

When an underscore is placed in this space it will be saved: ANCIENT\_GREECE.

---

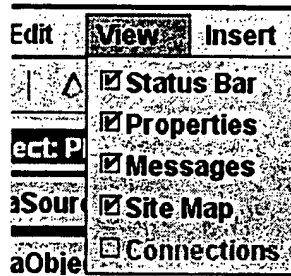
## *Access Flexibility*

**AltoStudio** has been designed to allow several approaches to accomplish the same task. You may, for instance, use either the menu bar or an icon on the Quick Access tool bar.

Additionally, the position of all dialog boxes can be changed by clicking and dragging them to any other position on the screen. They can also be minimized or expanded using the buttons in the upper right hand corner.

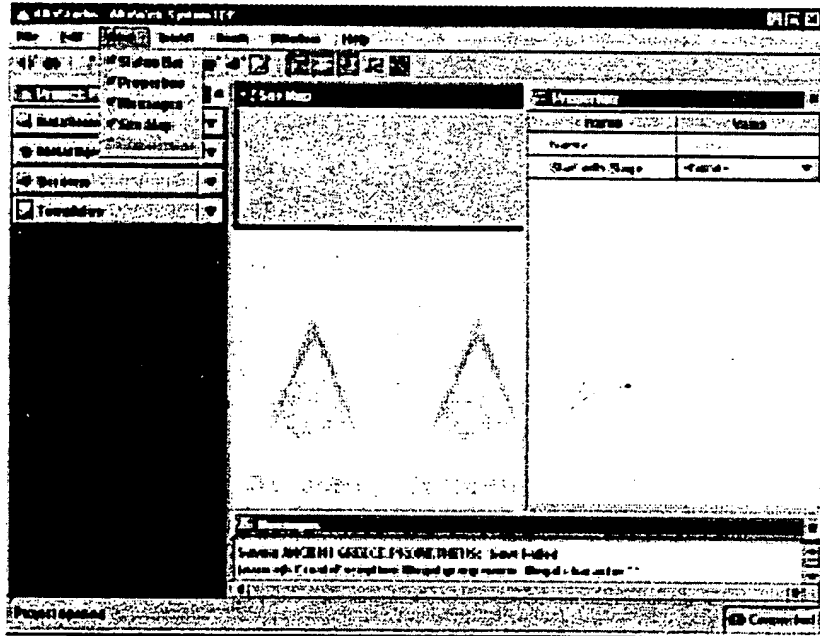
For example, the four buttons on the far right hand of the Quick Access Icon tool bar effectively duplicate the functionality of the menu bar field **AltoStudio > View**.

Specifically, placing a checkmark in each checkbox in the drop-down menu **File > View** produces the same dialog box as the associated icons.



In short, user preference and ease-of-use determine the choice whether you are going to use the menu bar or the tool bar.

Additionally, in the example shown below, you have the ability to enlarge or shrink each window as well as move the windows to any location on the screen.



Now that you are familiar with the concepts and flexible interrelationships of the three major components that constitute AltoWeb (including the menu bar and Quick Access icons), it is time to connect a server then create a project.

Chapter 3, "Connecting to a Server" on page 39, begins this process.



---

Before you can create a project, you must have a connection to a server.

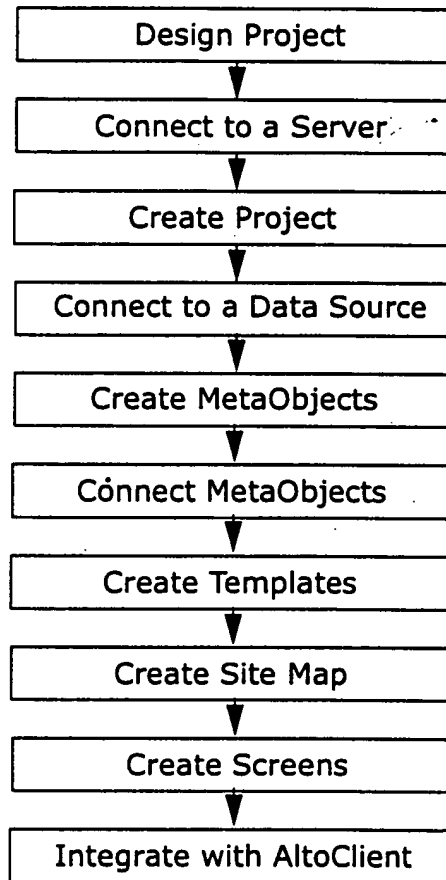
This chapter includes the following topics:

- "Flow Diagram" on page 40
- "Initializing AltoStudio" on page 41
- "Starting AltoStudio" on page 42
- "Connecting to a Server" on page 43

## *Flow Diagram*

The process of developing your project that will ultimately be reflected in the AltoClient begins with connecting to a server. After you have connected to a server you will create a project (or projects), create and map MetaObjects, then incorporate these MetaObjects within templates and site maps.

This process can be summarized through the following suggested flow chart.





This processes of this suggested block diagram flow chart are not fixed. You can proceed to your ultimate goal using any flow you find appropriate to your specific environment although certain events must be sequential, e.g., you must have a connection to a server before you can create a project and before you can designate your datasource.

On the other hand, you may find that once you have designed and created a project, you might prefer to create a site map and then proceed to connecting a data source. In short, there are alternative ways to develop your solution.

---

### *Initializing AltoStudio*

Before you begin, you must have the appropriate operating system (OS) software and hardware installed (See "System Requirements" on page 20.).

You must also have your AltoServer software installed and configured before running AltoStudio. You are referred to AltoServer documentation for this process.

This section covers the background information that allows you to begin the access of AltoWeb's functionality. It contains step-by-step procedures about how to use AltoStudio and, has been stated, does not require the knowledge of software languages to implement these procedures.

---

## Starting AltoStudio

The development of a project begins with the default file system window *runStudio.bat*. This file system was created in AltoServer.

### *runStudio.bat* Window

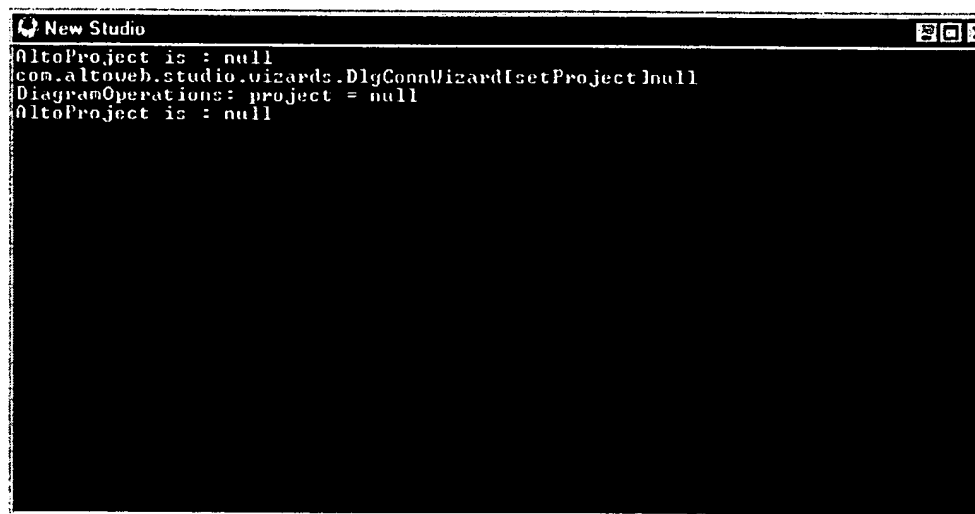
To access *runStudio.bat*:

1. Launch the Explorer window
2. Select the file *runStudio.bat*

If you want to install AltoStudio into a different file system, you must set up a link to the new location before starting the installation.

3. Double-click *runStudio.bat*

The *runStudio.bat* window opens.



```
New Studio
AltoProject is : null
com.altoweb.studio.wizards.DlgConnWizard[setProject]null
DiagramOperations: project = null
AltoProject is : null
```

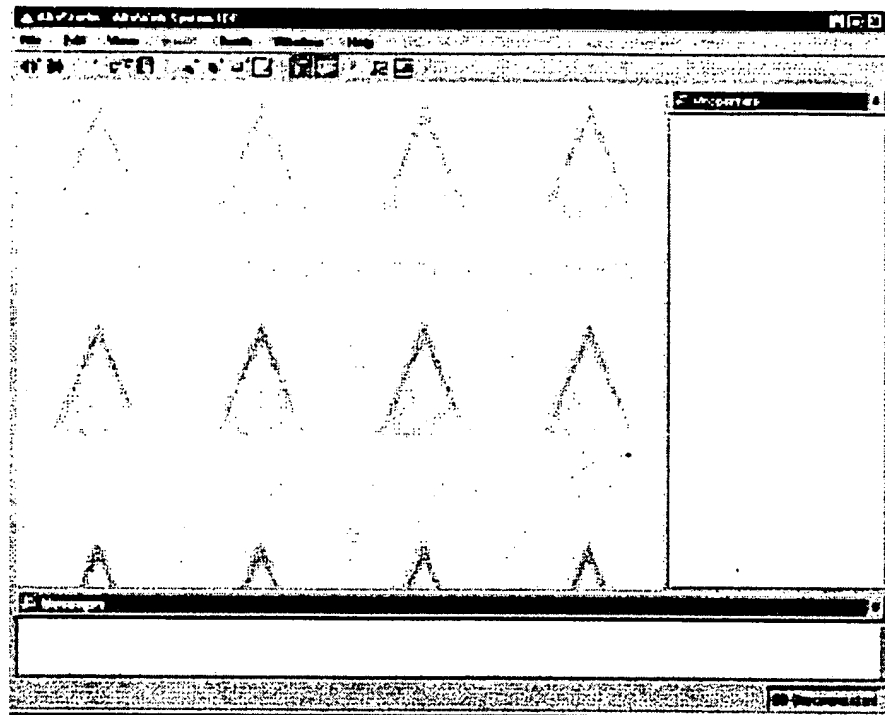
4. Type the file name in the Command Line.
5. Press **Enter**.

---

## Connecting to a Server

---

The AltoStudio main screen opens.



---

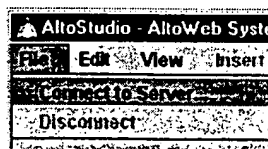
## Connecting to a Server

---

The first thing you must do is to connect to a server.

From the menu bar:

1. Highlight **File > Connect to Server**.



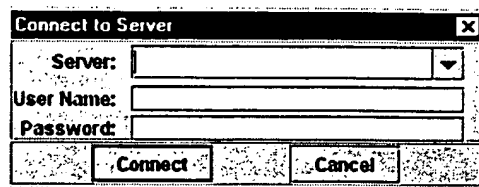
NOTE: You could also use the Quick Access Connect to Server



icon.

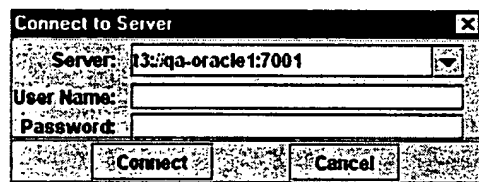
2. Click **Connect to Server**.

The **Connect to Server** dialog box appears.

A screenshot of the 'Connect to Server' dialog box. It has a title bar with a close button. Inside, there are three text input fields: 'Server:', 'User Name:', and 'Password:'. The 'Server:' field has a dropdown arrow on its right. At the bottom, there are two buttons: 'Connect' and 'Cancel'.

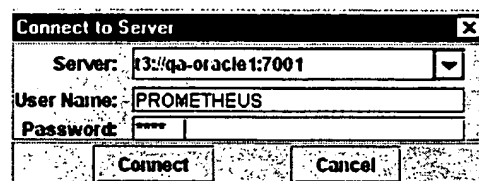
3. Enter the name and location of the server you want to use in the Server field.

A successful connection to a server is noted below. In this example, a server identified as `t3://qa-oracle1:7001` is used.

A screenshot of the 'Connect to Server' dialog box. The 'Server:' field now contains the text 't3://qa-oracle1:7001'. The 'User Name:' and 'Password:' fields are empty. The 'Connect' and 'Cancel' buttons are at the bottom.

NOTE: In the connection URL `T3://qa-oracle1:7001`, T3 represents an application server protocol supported by WebLogic® BEA systems®. It is not necessary to type `T3://` in your URL.

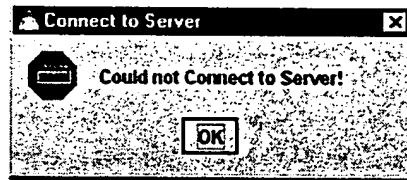
4. Enter your **User Name** and **Password** in their respective fields.

A screenshot of the 'Connect to Server' dialog box. The 'Server:' field contains 't3://qa-oracle1:7001'. The 'User Name:' field contains 'PROMETHEUS'. The 'Password:' field contains four asterisks '\*\*\*\*'. The 'Connect' and 'Cancel' buttons are at the bottom.

5. Press **Connect**.

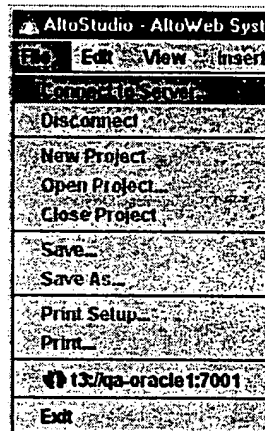
**NOTE:** Pressing the **Esc** key will stop your attempt at connecting to a server.

If the connection is not successful you will receive an error message in the Connect to Server dialog box: Could not Connect to Server!



6. Press **OK**.

Once the connection is successful, the server you have connected to is highlighted in the **AltoStudio > File** dialog box drop-down list.



You will also see the location of any previously connected servers as well as a list of projects you have worked on. You can highlight and click on any of these listed servers and projects to access it.

The status bar dialog field, located in the bottom right hand corner of the AltoStudio window, has changed from Disconnected to Connected. This indicates that the connectivity to the server is valid.



You are now ready to create a project.

Chapter 4, "Creating a Project" on page 47, tells you how to make this so.

## *Creating a Project*

---

Once you have determined what your project might look like, who is to use it, and where it is to be used, the process of implementing your project can begin.


---

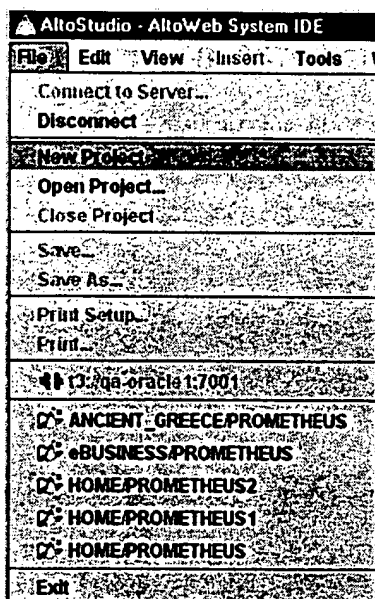
### *Creating a Project*

After you have connected to a server, creating a Project begins from the AltoStudio Window. You may create a new project, or open an existing one, as a step towards developing your project.

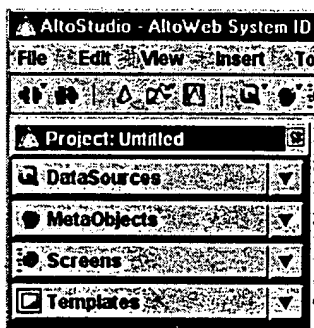
#### **New Project**

1. Highlight **File**
2. From the drop-down list, highlight **New Project**
3. Left click your mouse button

NOTE: You could also use the Quick Access New Project  icon.



An untitled project dialog box appears:





Regardless of the information you will adding to your project, it will remain untitled. A title will not be assigned until you save the information from DataSources.

At this point you can provide information to each of the four fields:

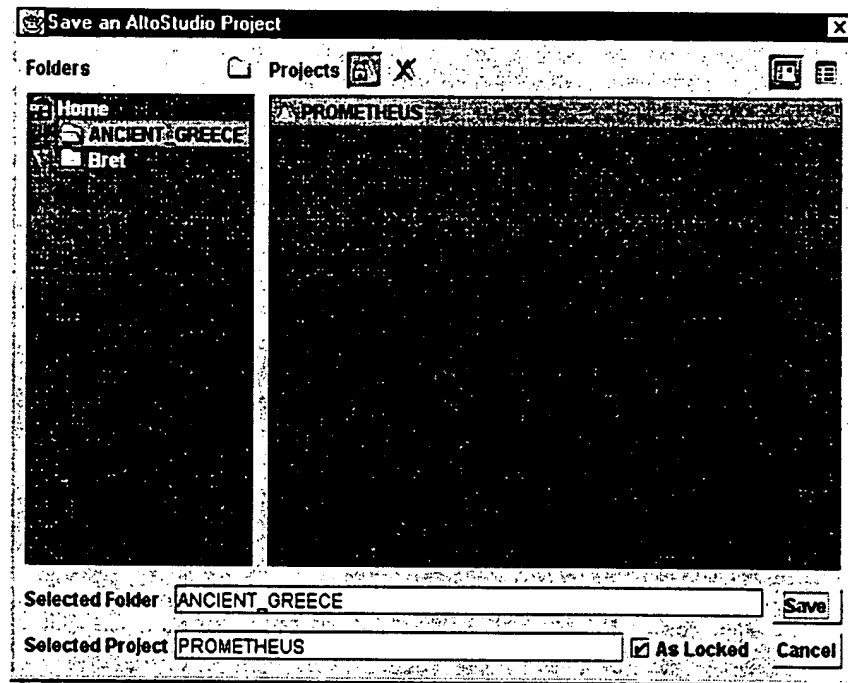
- DataSources (See "Connecting to a Data Source" on page 55.)
- MetaObjects (See "MetaObjects" on page 71.)
- Screens (See "Templates and Screens" on page 109.)
- Templates (See "Templates and Screens" on page 109.)

After you have determined your DataSource, you can name your project.

This is initiated through either **File > Save** or the Save Project

Icon .

Either method will open the **Save an AltoStudio Project** window.



Enter the name of your folder in the Selected Folder field and the name you want to use for your project in the Selected Project field.

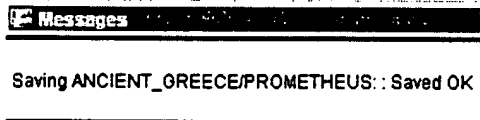
You can also use the icons in the menu bar of this screen to open a new project or delete an existing project. The two icons at the right allow you to see or reorder the listing of the contents of your project.



A check-mark in the check-box identified as *As Locked* means that subsequent users of your project will be able to access it, but not change it. You will essentially create a "read-only" project.

Press **Save**.


You will find that the Selected Project name you have entered is the new identity of the project and is reflected in the message window.



You can now populate your Project with MetaObjects.

## Open Project

If you do not want to create a new project, and you have saved an existing project, then click **File > Open Project**. This will access the Open an AltoStudio Project window.

**NOTE:** You could also use the Quick Access Open Project  icon.

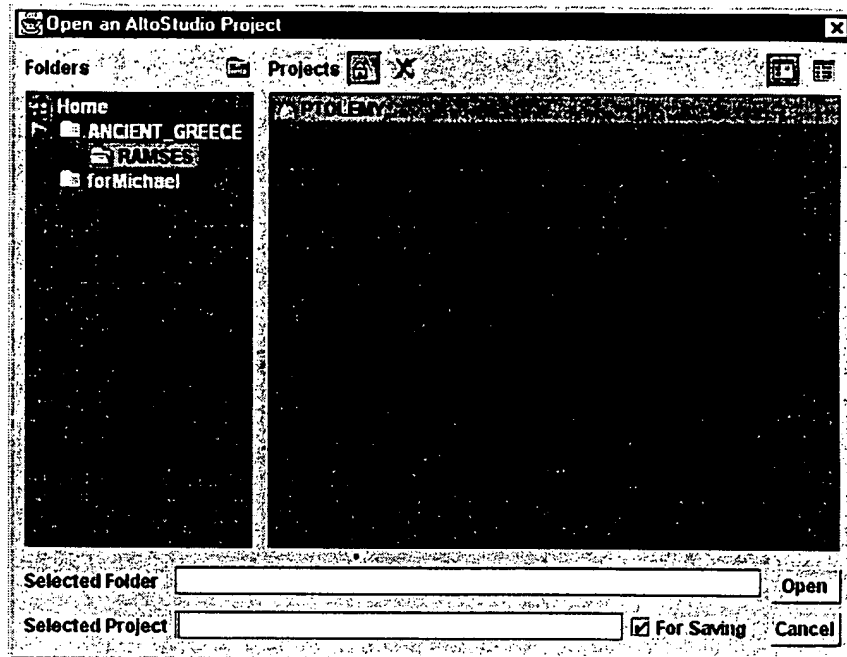
When you click and hold the **Open Project** field, the drop-down list will show the default maximum number of projects. This number is presently five projects.

If you want to set the maximum number of projects to greater than the default then:

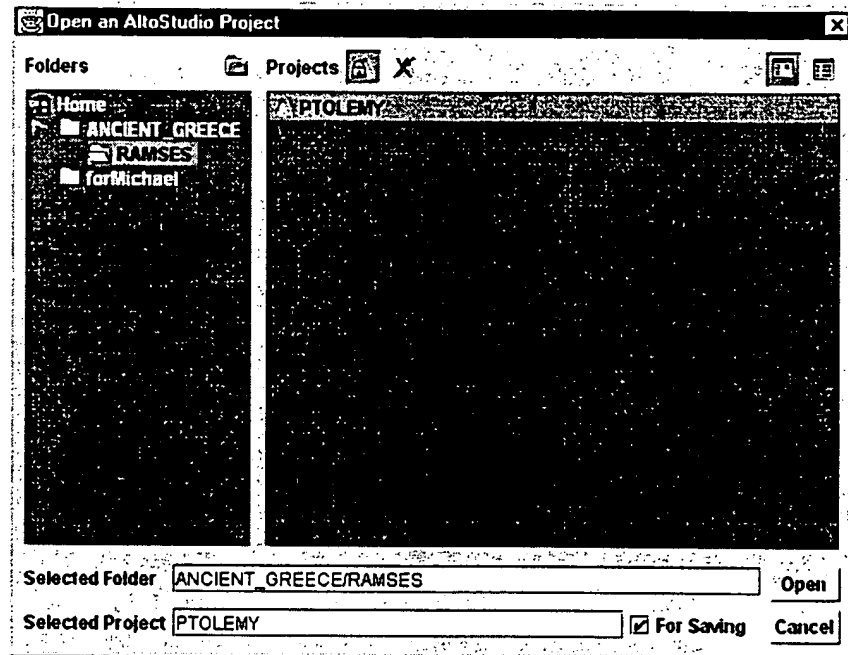
1. Start AltoStudio
2. Go to **Tools > Environment Options**
3. Enter the number of projects you want to see in the "max no of projects" field
4. Click **Enter**

**NOTE:** If you are going to change the default number of projects, you must do so **BEFORE** you open/create/save ANY AltoStudio Project. This because the "max no of projects" will start taking effect every time you open/save a project.

Given the above information, either five or more projects are shown in the **Open an AltoStudio Project** window. If there were no existing projects the contents of this window would, of course, be blank.



Since you are going to use an existing project, highlight the folder (ANCIENT\_GREECE/RAMSES) and project name (PTOLEMY).



These will then appear in the Selected Folder and Selected Project fields respectively.

Press **Open**.

Make sure that you save your Project before you proceed further. There are two ways to save your Project: either with the Quick

Access Save Project icon  or **File > Save Project**.

Now that you have created a new project, or opened an existing project, you must access a data source.

Chapter 4, "Connecting to a Data Source" on page 55, explains this process.

Copyright © 2008

---

At this point, you have created a project or opened an existing project. You will now associate your project with a data source.

This chapter includes the following topics:

- "Connecting to a Data Source" on page 56
- "New Project Data Source" on page 56
- "WebLogic Properties File" on page 63
- "Inserting a New DataSource" on page 66
- "DataSource Wizard" on page 67

---

## *Connecting to a Data Source*

---

Data sources are determined by, and come directly from, the server.

There are 3 distinct and separate ways to connect to a data source:

1. From the menu bar:  
**File > New Project > Data Sources**
2. From the menu bar:  
**Insert > New Data Source**
3. From the Quick Access Icon tool bar:  
**Data Source Wizard**

Each of these procedures will be presented in the following discussions.

---

## *New Project Data Source*

---

From the menu bar access **File > New Project.**

The Project drop-down dialog box appears.



Highlight **DataSources.**



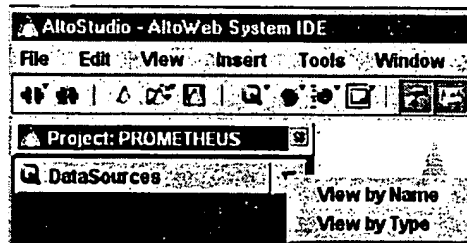
---

## New Project Data Source

---

Click the drop-down arrow in the right side of the DataSources field. This will bring up a dialog box with two choices.

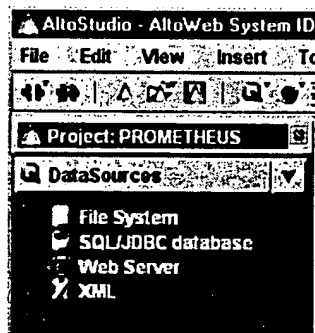
- View by Name
- View by Type



If you click View by Name you will see all of your data sources listed by their name. If you click the View by Type field, you will have a selection of alternative data sources by type. In either case, the selection of sources is determined by the AltoServer.

Presently, there are four types of data sources:

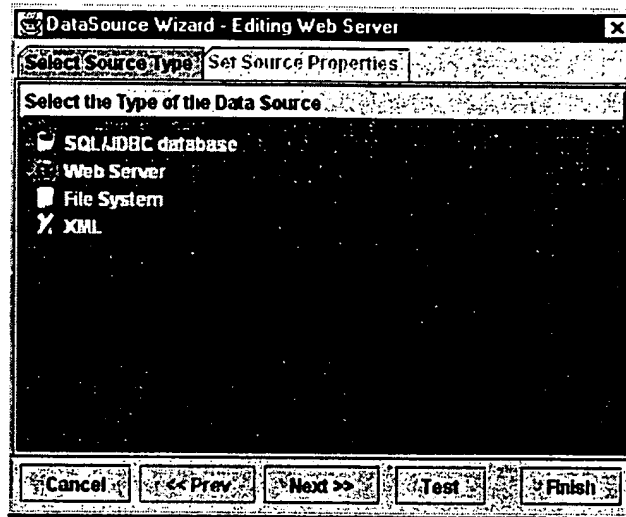
- File system
- SQL/JDBC database
- Web Server
- XML



Select any of the four available sources by highlighting it and left clicking your mouse button. This will automatically open the Data-Source Wizard dialog box.

## DataSource Wizard

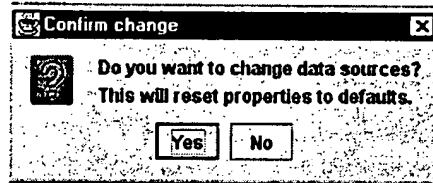
The **Data Source Wizard** window allows you to actually select a data source and configure the properties of that data source.



When the wizard opens, the Select Source Type tab is highlighted. The four fields within this tab are the same data sources that were available to you from the project menu bar.

For this example, we highlight the SQL/JDBC database although any source would be appropriate. The advantage of using the SQL/JDBC database is that there is an existing source of files which can be accessed.

**NOTE:** If you are not creating a new project, but opening an existing one, you may see a **Confirm change** dialog box. This dialog box appears if you are going to use a different data source than the one you previously selected.



Select, and press, the appropriate button (Yes or No) to confirm your preference. This will return you to the DataSource Wizard screen.

Press **Next**.

The Set Source Properties tab opens.

In this sample dialog box, the Set Source Properties tab is highlighted and the Selected Data Source Type tab Source Name is **SQL/JDBC database**.

The fields in the DataSource Wizard dialog box will differ in accordance with the Source Type that is selected. For example, the Web Server dialog box is substantially different than the SQL/JDBC dialog box.

Since SQL/JDBC was selected in this example, User name, Password, JDBC driver and Database URL are required fields.

DataSource Wizard - New DataSource

Select Source Type **Select Source Properties**

Selected Data Source Type: SQL/JDBC database

Source Name: SQL/JDBC database

User name:

Database URL: jdbc:weblogic:oracle:theDatabaseUrl

JDBC driver: weblogic.jdbc.oci.Driver

Password:

Connect Options:

☒ Save Password?

Cancel << Prev Next >> Test Finish

Your database administrator should provide the information that you use in this dialog box. That is, the information in the open fields are associated with a conventional Oracle connection pool database which has been created in WebLogic properties (See "WebLogic Properties File" on page 63.). This is the domain of the database administrator.

The following information is required:

- User name
- Password
- Connect Options (not required except in special situations by the database administrator)

In this example the user name is `scott` and the password is a five digit name noted by asterisks (these will be the same whether it is SQL or ISQL).

NOTE: Entries in these fields are case sensitive.

DataSource Wizard - New DataSource

Select Source Type | **Set Source Properties**

Selected Data Source Type: SQLJDBC database

Source Name: SQLJDBC database

User name: scott

Database URL: jdbc:weblogic:oracle:qa-oracle1

JDBC driver: weblogic.jdbc.oci.Driver

Password:

Connect Options:

☒ Save Password ?

Cancel << Prev Next >> **Test** Finish

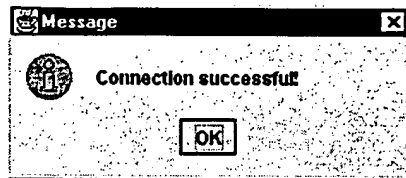
Press **Test** to confirm your connectivity.

## Test

The **Test** button allows you to determine whether the information you have entered in the open fields is sufficient to represent a valid connection. If, for any reason, the system cannot establish a valid connection for any of these fields, it will generate an error message which must be resolved before you can progress further.

If, for any reason, you cannot determine that you have a valid connection, you can either continue with that data source or change your source using the Select Source Type tab. Sometimes, selecting a different source will resolve connectivity issues.

Once you have verified a connection to your data source, you will receive a Connection successful message.



Press **OK**. This will return you to the DataSource Wizard.

From the DataSource Wizard window, press **Finish**.

---

## *WebLogic Properties File*

You cannot run your WebLogic Server without setting certain required configuration properties in your Java utilities WebLogic Properties file.

Although the WebLogic Properties file is normally the domain of your database administrator or system administrator, it is briefly presented here for informational purposes.

Your WebLogic Properties file is accessed through your WordPad:

**Start > Programs > WebLogic > Utilities > Edit Server Properties**

[illegible][illegible]


—



---

## Data Source Properties

If you want to check the properties of an existing data source — or the properties of any other field item for that matter — then highlight the source and either right-click your mouse or click Properties.

You could also double-click the Properties  icon.

A typical Properties dialog box would resemble the one shown below. In this example, the dialog is for a Web Server data source, not a SQL/JDBC, and the URL that was chosen is yahoo.

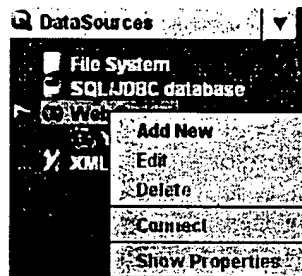
Properties	
Name	Value
Name	YAHOO
TypeName	Web Server
Source Specific	
URL	http://www.yahoo.com/

If you want to rename the Web Server, you only have to highlight the Name Value field and enter your change. In this instance we changed the Value from Web Server to YAHOO. You are allowed to change nomenclature information within any active field in a dialog box (active fields are not gray).

An active field is uses normal type face whereas an inactive field uses a gray type face. In this dialog box, Name can be changed but TypeName cannot. In short, you can edit Name but not Type-Name since the TypeName represents a path to the actual data-base source.

## Modifying a DataSource

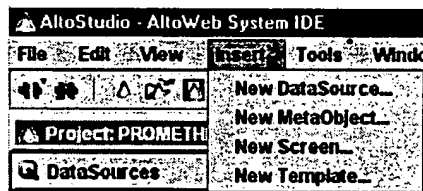
Highlight any of the four DataSources and right-click your mouse to bring up the following dialog box. This dialog box can be used to Add New, Edit, Delete, Connect, or Show Properties of the data source.



---

## Inserting a New DataSource

From the menu bar, highlight and double-click **Insert > New DataSource**.




This will access the DataSource Wizard.

---

## *DataSource Wizard*

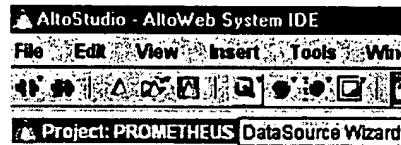
---

The DataSource Wizard can be accessed in three different ways:

1. From the tool bar menu **File > New Project > Project > DataSources> (source)**.
2. From the **Insert > New DataSource**
3. From the Quick Access tool bar  DataSource Wizard icon.

While you could use the menu bar most people prefer the DataSource Wizard icon because of the ease of use and speed. Additionally, many users prefer to use icon technology rather than files.

This section will only discuss icon functionality.



Regardless of the path you have chosen to access the wizard, you will access the same DataSource Wizard dialog box.

An additional advantage of using an DataSource Wizard icon rather than the menu bar is that any existing information in the wizard that has been previously associated with your source will be shown. It will not have to be reentered.

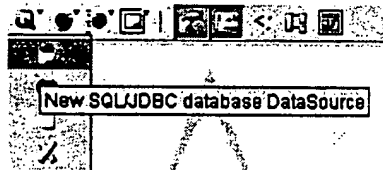
As the functionality of the DataSource Wizard using the menu bar was previously discussed in "New Project Data Source" on page 56, it will not be duplicated here.

## Drop-down Alternatives

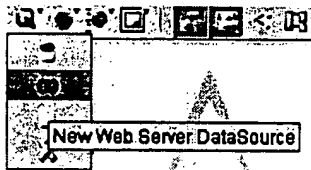
There are four drop-down alternatives from the Quick Access DataSource Wizard icon. These alternatives duplicate the alternatives found in the Source Type selection area.

Click and hold your cursor on the DataSource Wizard icon to access the list of available source types. Clicking these buttons will immediately select that source on the wizard screen.

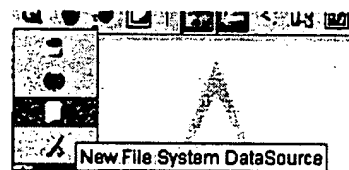
- New SQL/JDBC database DataSource



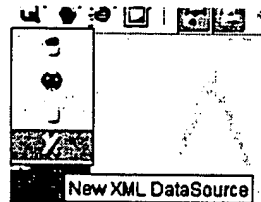
- New Web Server DataSource



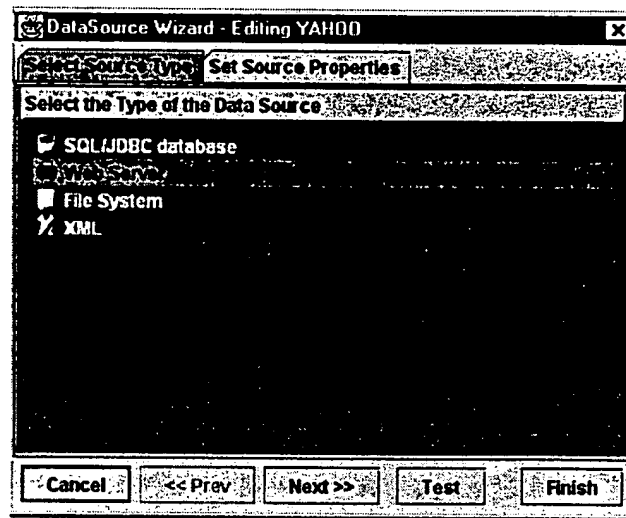
- New File System DataSource



- New XML DataSource



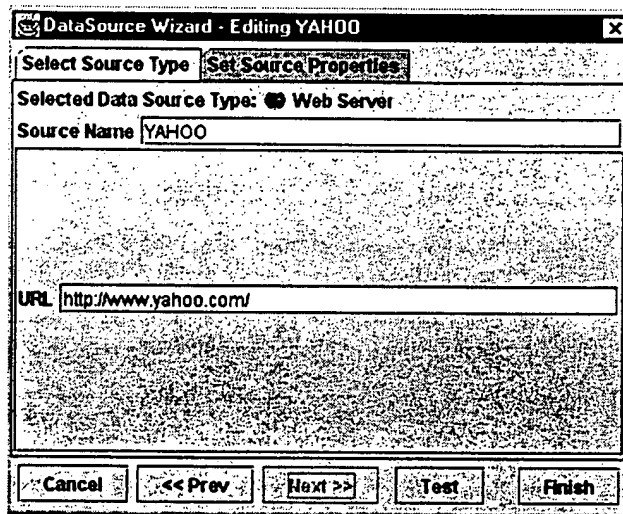
For example, you can add a new Web Server DataSource by clicking the New Web Server DataSource icon. This will access the DataSource Wizard's Select Source Type tab where you would Highlight Web Server.



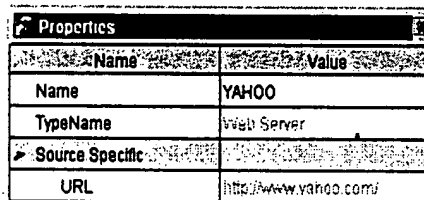
In this instance, you will note that the DataSource Value Name was changed from Web Source to YAHOO in the Properties pane (See "Data Source Properties" on page 65.) and is reflected in the banner.

Click **Next**.

The Set Source Properties tab opens. You can now set the new Source Name to reflect the URL `www.yahoo.com`.



This can also be confirmed by the Properties dialog box.



Click **Test** if you want to confirm the validity of your connection.

Click **Finish**.

Now that you have created and named a project (or projects) and connected them to a server and a data source, you are ready to create MetaObjects.

Chapter 6, "MetaObjects" on page 71, tells you how this process can be developed.

---

At this point, you have created a project, PROMETHEUS, linked it to a server identified as t3://qa-oracle1:7001, and associated it with a Web Server Data Source.

You are now ready to create the MetaObjects. MetaObjects are used to populate your project.

The purpose of this chapter is to discuss how to create MetaObjects. The following chapter — Chapter 7, “Connecting MetaObjects” on page 101 — will show you how to form relationships between those MetaObjects.

This chapter includes the following topics:

- “Background” on page 72
- “MetaDefinitions” on page 72
- “Creating MetaObjects” on page 73
- “Insert > New MetaObject” on page 74
- “MetaObject Wizard Icon” on page 97

---

## *Background*

A MetaObject contains user-defined information or links that are used to eventually provide the content that is used to populate the AltoClient interface.

AltoStudio has the ability to:

- Create MetaObjects using information tables from your data source
- Map, or connect, MetaObjects to other MetaObjects

---

## *MetaDefinitions*

A MetaObject and a MetaObject Connection can be defined as:

### **MetaObject**

At a fundamental level, a MetaObject is a description of data. At a more explicit level, a MetaObject is a concept that represents a homogeneous description of data within the AltoWeb system; a description that is used as a representation from heterogeneous data systems.

A MetaObject may also contain fields from a homogeneous data source including, for instance, information in a table or hyperlinks in an HTML document.

A MetaObject might consist of a selection of all automobiles where:

*location = germany*

Or all hyperlinks that start with:

*www.gullwing.com*



MetaObjects can also have additional constraints including, for example:

*automobiles = gran touring = german = gullwing*

### MetaObject Connection

A relationship between MetaObjects is known as a MetaObject Connection. It is used to define on-screen states on the sphere or cube. It is a "drill-down" that is defined by the MetaObject Connection Query tab (See "Define Queries Tab" on page 90.).

In short, a MetaObject Connection is used to define links between objects.

---

## Creating MetaObjects

After you have logged onto the appropriate server and established a connection to a data source you are ready to create a MetaObject.

**NOTE:** While a data source is not required to create a MetaObject, it is the normal condition.

There are two ways to create a new MetaObject:

- From the Quick Access tool bar MetaObject Wizard icon
- From the menu bar: **Insert > New MetaObject**

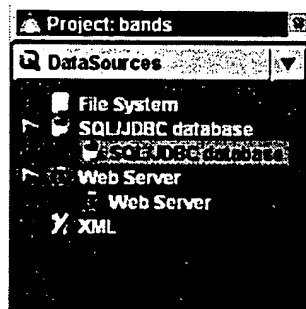
Both protocols employ the use of the MetaObject Wizard (See "MetaObject Wizard" on page 75.).

---

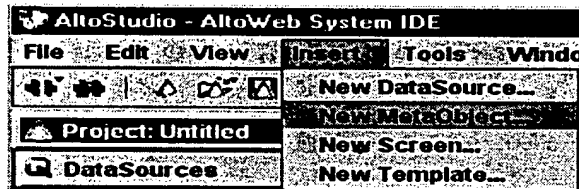
## *Insert > New MetaObject*

---

Since we have created a project and selected an SQL/JDBC data-  
base in Chapter 5, we have a source of files (See “DataSource  
Wizard” on page 59.) which can be used as a basis for the cre-  
ation of a MetaObject.



From the AltoStudio window menu bar, highlight and click  
**Insert > New MetaObject.**



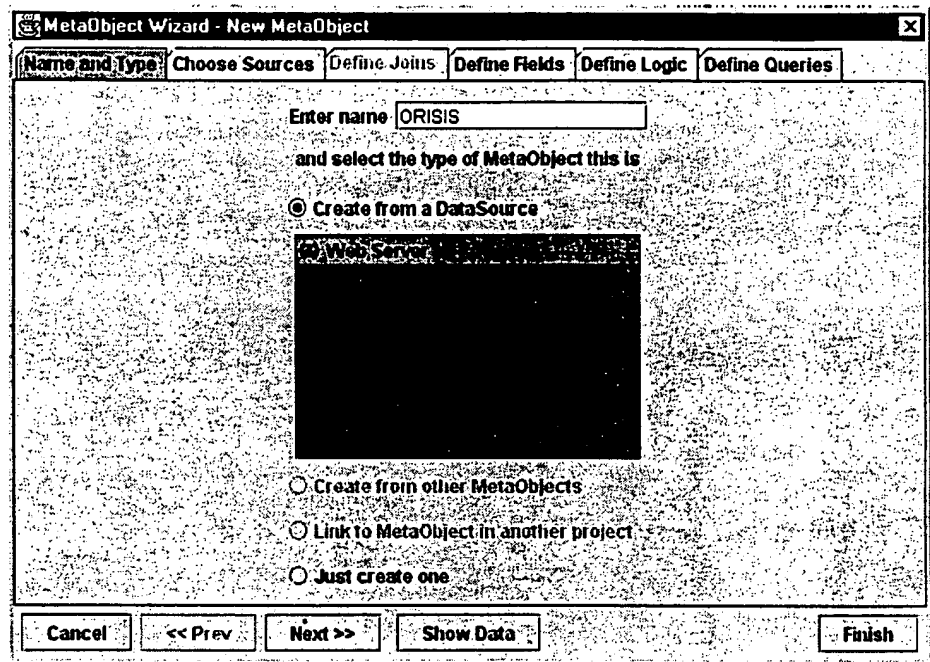
Clicking New MetaObject displays the MetaObject Wizard.

---

## MetaObject Wizard

---

The MetaObject Wizard is used to create and define MetaObjects. For this example, we are going to change the data source from SQL/JDBC to the Web Server. The only purpose is to show the flexibility of this system.



We have named this MetaObject ORISIS and highlighted Web Server as the database. You can, of course, choose your own nomenclature as well as data source. For example, in the subsequent examples, the data source was SQL/JDBC. Regardless of the source, the final result will reflect the needs of your project.

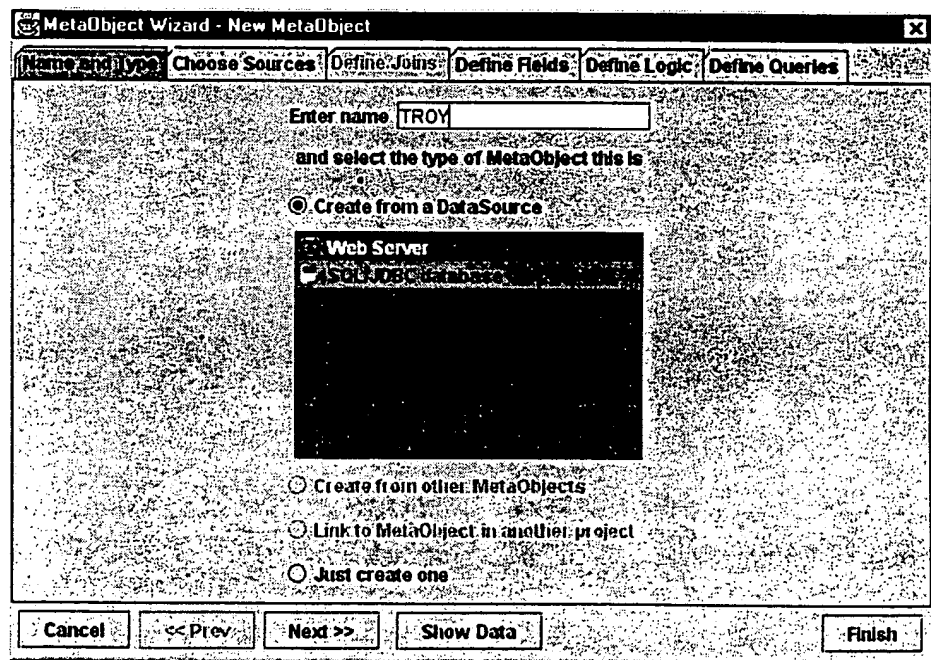
There are six tabs in the MetaObject Wizard window.

- Name and Type
- Choose Sources
- Define Joins
- Define fields
- Define Logic
- Define Queries

### Name and Type tab

Accessing the MetaObject Wizard window opens the **Name and Type** tab.

Use the *Enter name* field to create, or change, a name for your MetaObject. If you place a check-mark in the *Create from a DataSource* radio button then your MetaObject will be derived from an existing DataSource where that name is located.



In this instance we are using the **Name and Type** tab to change the data source and identify a MetaObject named TROY created from a SQL/JDBC database.

The next two radio buttons: *Create from other MetaObjects* and *Link to MetaObject in another project* are associated with the functionality of the Compound MetaObject and Linked MetaObject respectively, found in the Quick Access MetaObject Wizard icon drop-down list (See "MetaObject Wizard Icon" on page 97.).

The *Just create one* radio button means that you are creating a MetaObject without referencing it to a DataSource.

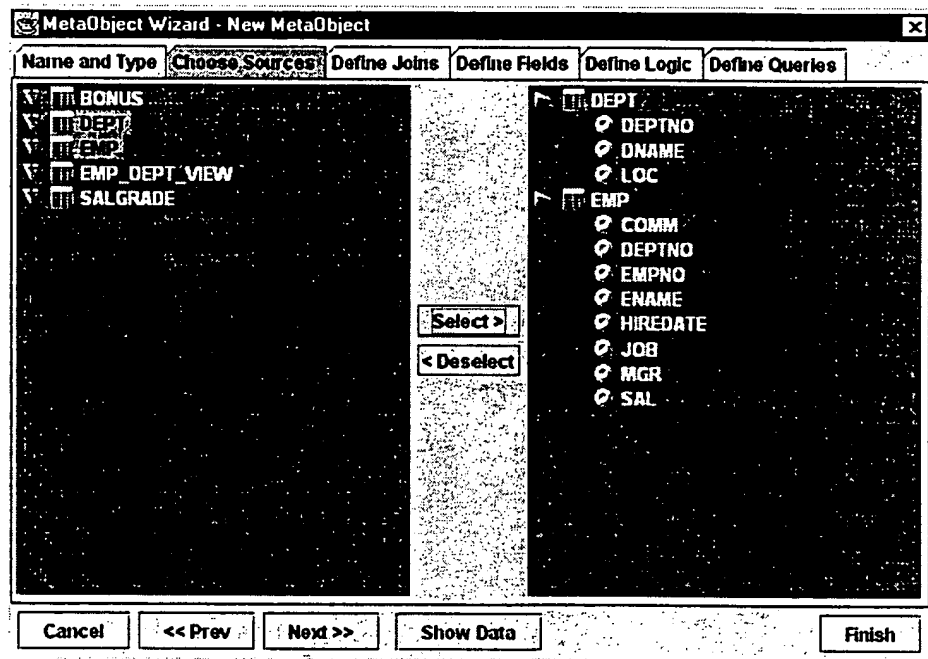
Press **Next**.

The Choose Sources tab opens.

## Choose Sources tab

MetaObjects are defined from data sets using the Choose Sources tab.

The Choose Sources tab allows you to select specific tables and data sets from your data source (HTML DataSet in this example) to include in the MetaObject.



The left pane in the MetaObject Wizard shows the fields and data sets which are available from the SQL/JDBC data source.

If, however, you placed a mark in the Create from other MetaObjects radio button in the Name and Type tab, then these fields and data sets would be replaced by MetaObjects.

In this example the table, HTMLDataSet, is highlighted.

Click the **Select >** button to transfer DEPT and EMP information to the right pane.

If you do not need any of the fields, then highlight it in the right pane and click **< Deselect**. This removes that field from the right pane and returns it to the original, left pane. This process can be used for any field or table that you want to be either selected or deselected.

Press **Next**. The Define Joins tab opens.

## Define Joins tab

The Define Joins tab is used to decide how you want to link diverse data sets of MetaObjects. If it is XML orientated, it requires a master/child relationship.

The Define Joins tab is used when you have more than one data set that is going to be used concurrently. In this instance, you designate one of the data sets as a master and the other as a child.

The screenshot shows the 'MetaObject Wizard - New MetaObject' dialog box with the 'Define Joins' tab selected. The dialog has a tabbed interface with the following tabs: 'Name and Type', 'Choose Sources', 'Define Joins' (active), 'Define Fields', 'Define Logic', and 'Define Queries'. Below the tabs, there is a dropdown menu labeled 'Select the Root DataSet for Joins'. Underneath this is a section titled 'Current Joins' which contains a table with four columns: 'Master DataSet', 'Master Field', 'Child DataSet', and 'Child Field'. The table is currently empty. Below the table, there are four dropdown menus: 'Master DataSet', 'Master Field', 'Child DataSet', and 'Child Field'. Below these are two more dropdown menus: 'Join Type' and 'Join Type'. At the bottom of the dialog, there are several buttons: 'Add', 'Modify', 'Clear', 'Remove', 'Cancel', '<< Prev', 'Next >>', 'Show Data', and 'Finish'.

Use the Data Sets you selected in the Choose Sources tab to determine the Master and Child relationships. The drop-down arrows provide the range of selections. In this instance, we selected DEPT as the Master DataSet and DEPTNO as the Master field. EMP is the Child DataSet and DEPTNO is the Child Field.

The screenshot shows the 'MetaObject Wizard - New MetaObject' dialog box, specifically the 'Define Joins' tab. The 'Name and Type' tab is selected. The 'Select the Root DataSet for Joins' dropdown is set to 'DEPT'. Below this, the 'Current Joins' table is empty. The 'Master DataSet' is 'DEPT' and the 'Master Field' is 'DEPTNO'. The 'Child DataSet' is 'EMP' and the 'Child Field' is 'DEPTNO'. The 'Join Type' is 'Only if fields match'. At the bottom, there are buttons for 'Add', 'Modify', 'Clear', 'Remove', 'Cancel', '<< Prev', 'Next >>', 'Show Data', and 'Finish'.

Master DataSet	Master Field	Child DataSet	Child Field
----------------	--------------	---------------	-------------

Master DataSet: DEPT Master Field: DEPTNO  
Child DataSet: EMP Child Field: DEPTNO  
Join Type: Only if fields match

Buttons: Add, Modify, Clear, Remove, Cancel, << Prev, Next >>, Show Data, Finish

Press **Add**.



This will show how your Master and Child DataSets and Fields are related.

The screenshot shows the 'MetaObject Wizard - New MetaObject' dialog box, specifically the 'Define Joins' tab. The 'Name and Type' tab is selected, showing 'DEPT' as the root Data Set. Below this, the 'Current Joins' table lists a join between 'DEPT' and 'EMP' on the 'DEPTNO' field. The 'Join Type' is set to 'Only if fields match'. At the bottom, there are buttons for 'Add', 'Modify', 'Clear', 'Remove', 'Cancel', '<< Prev', 'Next >>', 'Show Data', and 'Finish'.

Master DataSet	Master Field	Child DataSet	Child Field
DEPT	DEPTNO	EMP	DEPTNO

Master DataSet: DEPT Master Field: DEPTNO  
 Child DataSet: EMP Child Field: DEPTNO  
 Join Type: Only if fields match

Buttons: Add, Modify, Clear, Remove, Cancel, << Prev, Next >>, Show Data, Finish

Highlight the DEPT Master DataSet field and press **Show Data**.

This will show the data from which your fields are based.

Query Results for bands: TROY									
bands: TROY									
26 rows									
HIREDATE	DEPTNO	LOC	SAL	EMPNO	DEPTNO2	COMM	MGR	ENAME	
12-17-1980 12:00 AM	20	DALLAS	800.0	7369	20	1000.0	7902	SMITH	C
02-20-1981 12:00 AM	30	CHICAGO	1600.0	7499	30	300.0	7698	ALLEN	S
02-22-1981 12:00 AM	30	CHICAGO	1250.0	7521	30	500.0	7698	WARD	S
04-02-1981 12:00 AM	20	DALLAS	2975.0	7566	20	1000.0	7839	JONES	M
09-28-1981 12:00 AM	30	CHICAGO	1250.0	7654	30	1400.0	7698	MARTIN	S
05-01-1981 12:00 AM	30	CHICAGO	2850.0	7698	30		7839	BLAKE	M
06-09-1981 12:00 AM	10	NEW YORK	2450.0	7782	10	360.0	7839	CLARK	M
04-19-1987 12:00 AM	20	DALLAS	3000.0	7788	20	1000.0	7566	SCOTT	A
11-17-1981 12:00 AM	10	NEW YORK	5000.0	7839	10	360.0		KING	P
09-08-1981 12:00 AM	30	CHICAGO	1500.0	7844	30	0.0	7698	TURNER	S
05-23-1987 12:00 AM	20	DALLAS	1100.0	7876	20	1000.0	7788	ADAMS	C
12-03-1981 12:00 AM	30	CHICAGO	950.0	7900	30		7698	JAMES	C
12-03-1981 12:00 AM	20	DALLAS	3000.0	7902	20	1000.0	7566	FORD	A
01-23-1982 12:00 AM	10	NEW YORK	1300.0	7934	10	360.0	7782	MILLER	C
	50000	San Mateo	10600.0	2978	50000	5685.0	683	Tom Jones Jr	IT
	40000	San Jose	12000.0	2998	40000	6695.0	687	Dr Fu Man Chin	P
06-22-1999 12:00 AM	30000	Boston	16000.0	2678	30000	5475.0	693	Bill Me Later	O
	20000	California	96000.0	478	20000	5875.0	624	Angela is a Hottie	M
	10000	Los Angeles	17000.0	2278	10000	4275.0	603	Fred Stein	B
	60000	San Diego	14000.0	6779	60000	1075.0	402	Kathy Spencer	L

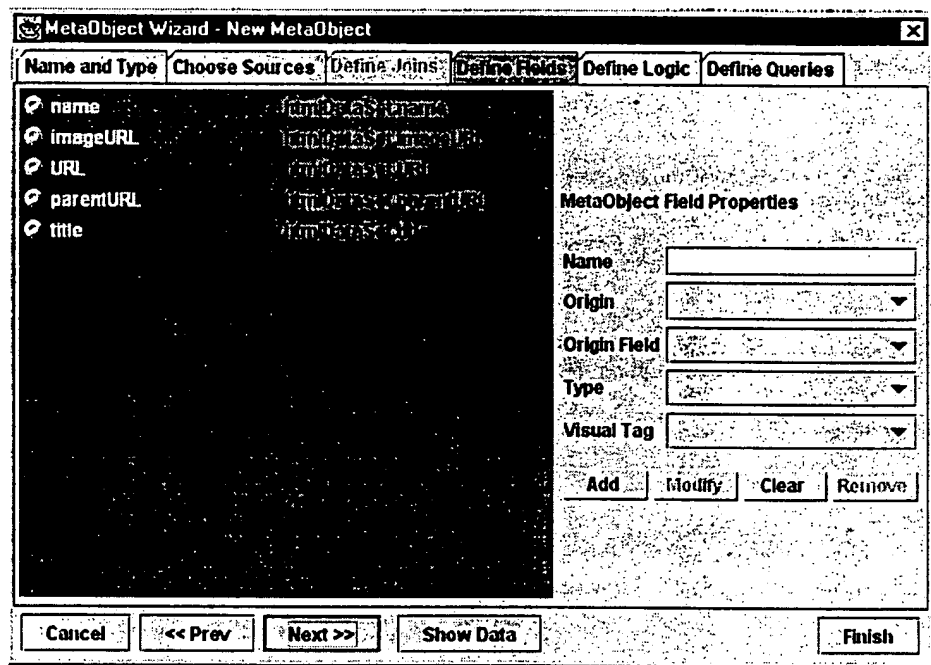
Press **Next**. This will open the Define Fields tab.

## Define Fields tab

The Define Fields tab uses information from the Choose Sources and Define Joins tabs to define specific MetaObject properties.

These properties are then used in the AltoClient window. The Define Fields tab can also be used to rename your preferences.

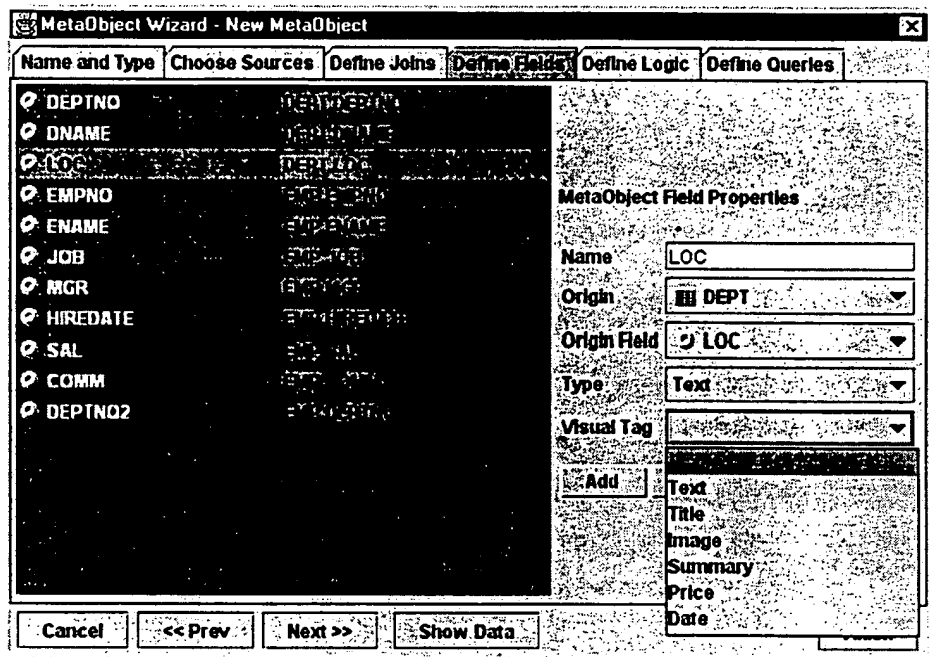
You can also use MetaObject Field Properties/Name field to rename the imported database field as well as select your Origin, Origin Field, Type, and Visual Tag.



In short, MetaObject Field Properties in the Define Fields tab are used to define the MetaObject characteristics which are used in the designated project.

Use the MetaObject Field Properties/Name field to add or define your field. This name will show as a field in the MetaObject in the **Connection Wizard > Project > MetaObjects** (See "Connection Wizard" on page 105.).

For example, if you highlight the field **LOC/DEPT.LOC** in the MetaObject Wizard window, the MetaObject Field Properties fields will reflect all of the source information contained in that field.



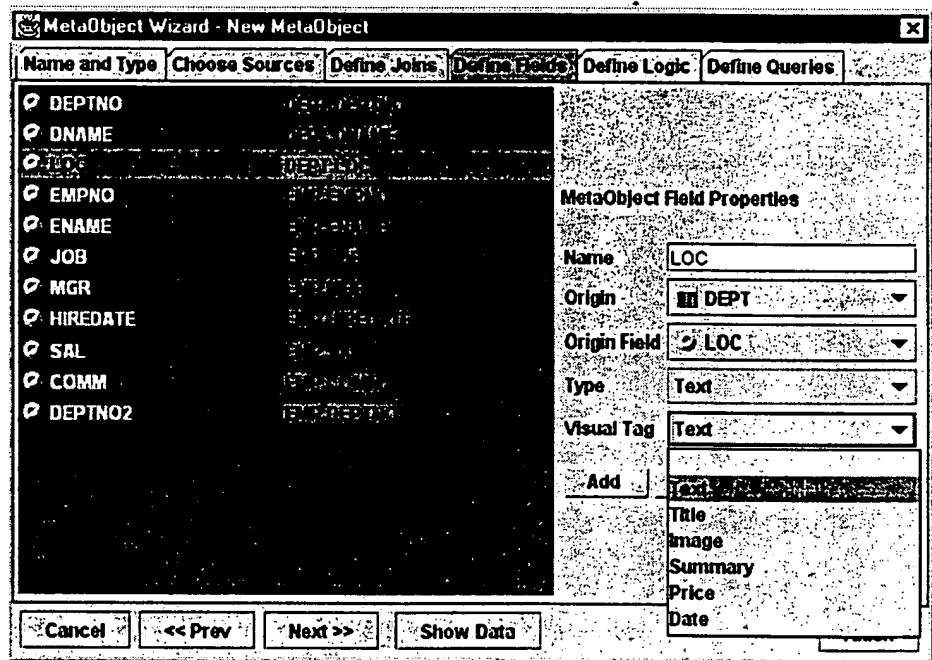
The Origin, Origin Field and type fields are used as information for the user. All of the properties in the MetaObject Field Properties dialog box can be modified in accordance with your design.

## Visual Tag

The Visual Tag field (See "Quick Access Add Visual Tabs Icon" on page 93.) is used to identify and position information on the facets. Visual Tag information is used with the template and context editors.

Without data, the facets and pop-ups will be automatically populated when you assign a field. Otherwise you have to drag-and-drop the data using the Screen Editor.

For example, if you wanted to identify the LOC/DEPT.LOC visual tag, you only have to click the drop-down arrow to find the alternatives and highlight your choice.



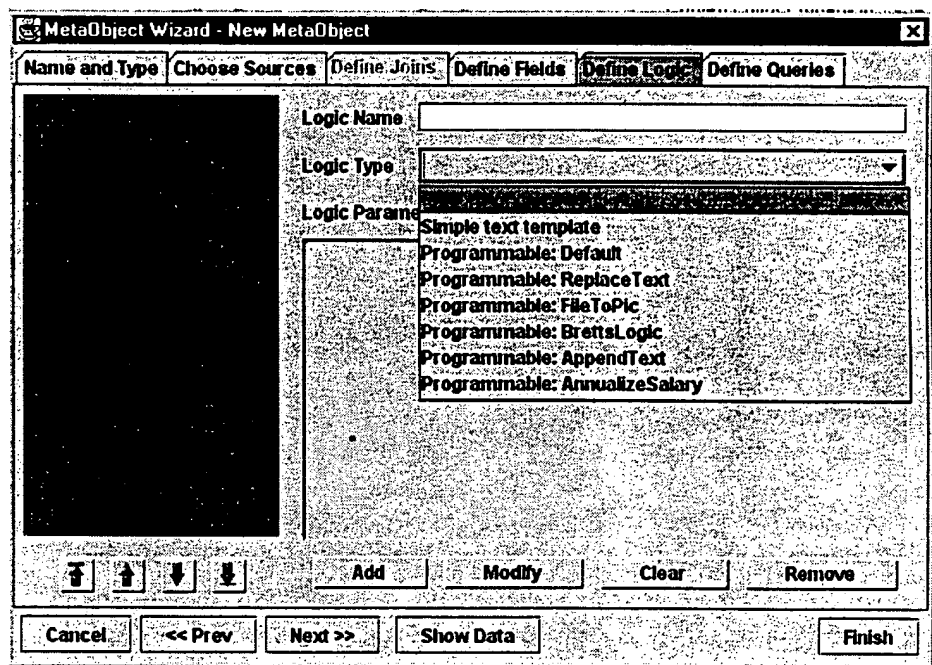
Click **Finish**.

## Define Logic Tab

The Define Logic Tab is used to apply business logic to existing fields.

While the Define Logic tab supports features that are used in the AltoStudio, its use is recommended for the advanced user.

In the illustration shown below, the Logic Type field drop-down arrow shows the several types of programmable logic which are available. Items on this list can be added to, modified, or deleted using the appropriate button on the bottom task bar.



In this example, if you want to change the name of a city from Chicago to Sacramento, then use the Logic Tab to locate and replace that geographical location.

The process is straightforward:

1. Enter a name in the **Logic Name** field: CHANGE LOCATION
2. Select a type of logic from the drop-down list and, by highlighting it, place in the **Logic Type** field: Programmable: Replace Text

MetaObject Wizard - New MetaObject

Define Logic

Logic Name: CHANGE LOCATION

Logic Type: Programmable: ReplaceText

Logic Parameters

Programmable: ReplaceText

Input field selection

Read Field: LOC

Output field selection

Write Field: LOC

Other parameters

Text to Replace: CHICAGO

Replacement Text: SACRAMENTO

Buttons: Add, Modify, Clear, Remove, Cancel, << Prev, Next >>, Show Data, Finish

3. In the **Input field selection/Read Field** enter the Origin Field: LOC
4. In the **Output Field Selection/Write Field** enter the Origin Field: LOC
5. In the **Other parameters/Text to Replace** field enter the geographical name of the city you want to change: CHICAGO
6. In the **Other parameters/Replacement Text** field enter the geographical name of the city you want to change to: SACRAMENTO

7. Click **Finish**.

You will note that this logic is now applied by the field highlighted in the left pane.

The screenshot shows the 'MetaObject Wizard - New MetaObject' dialog box, specifically the 'Define Logic' tab. The 'Logic Name' is 'CHANGE LOCATION'. The 'Logic Type' is 'Programmable: ReplaceText'. The 'Logic Parameters' section shows 'Input field selection' as 'LOC', 'Read Field' as 'LOC', 'Output field selection' as 'LOC', 'Write Field' as 'LOC', 'Text to Replace' as 'CHICAGO', and 'Replacement Text' as 'SACRAMENTO'. The 'Finish' button is highlighted.

Tab	Logic Name	Logic Type	Logic Parameters
Define Logic	CHANGE LOCATION	Programmable: ReplaceText	<p>Input field selection: LOC</p> <p>Read Field: LOC</p> <p>Output field selection: LOC</p> <p>Write Field: LOC</p> <p>Text to Replace: CHICAGO</p> <p>Replacement Text: SACRAMENTO</p>



Clicking the **Show Data** button will show the query results for your change.

Query Results for Table: EMP

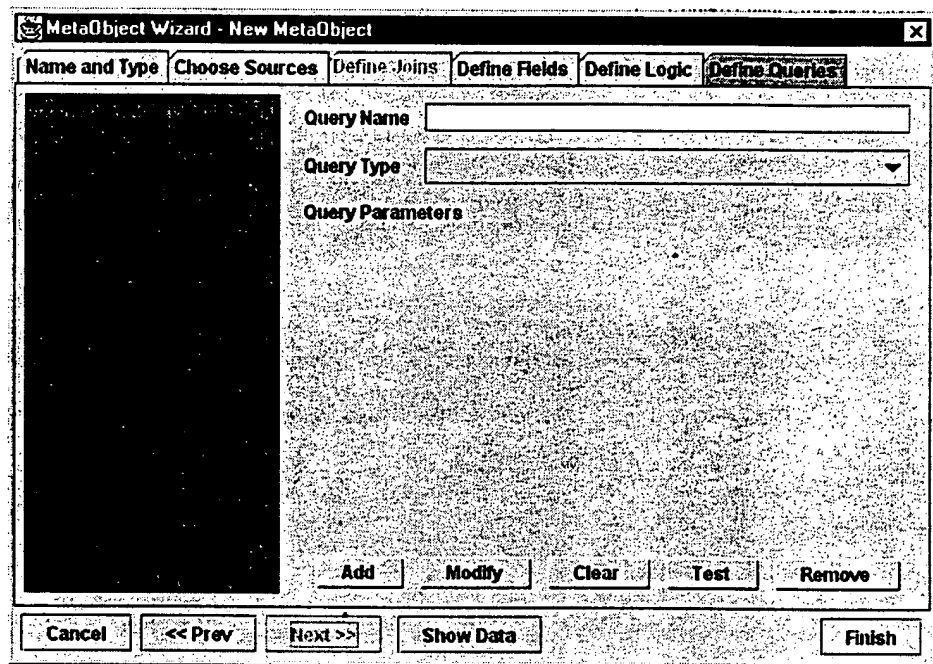
20 rows

HIREDATE	DEPTNO	LOC	SAL	EMPNO	DEPTNO	COMM	EMPNO	ENAME	JOB
09-22-1980	10	NEW YORK	10000.0	10000	10	0.0	10000	JOHN SMITH	MANAGER
09-22-1980	10	NEW YORK	5000.0	10001	10	0.0	10001	JAMES CLARK	CLERK
09-22-1980	10	NEW YORK	3000.0	10002	10	0.0	10002	ALLEN	SALESMAN
09-22-1980	10	NEW YORK	2000.0	10003	10	0.0	10003	WARD	SALESMAN
09-22-1980	10	NEW YORK	1000.0	10004	10	0.0	10004	MARTIN	SALESMAN
09-22-1980	10	NEW YORK	1000.0	10005	10	0.0	10005	BLAKE	MANAGER
09-22-1980	10	NEW YORK	1000.0	10006	10	0.0	10006	TURNER	SALESMAN
09-22-1980	10	NEW YORK	1000.0	10007	10	0.0	10007	JAMES	CLERK
09-22-1980	10	NEW YORK	1000.0	10008	10	0.0	10008	SCOTT	ANALYST
09-22-1980	10	NEW YORK	1000.0	10009	10	0.0	10009	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10010	10	0.0	10010	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10011	10	0.0	10011	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10012	10	0.0	10012	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10013	10	0.0	10013	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10014	10	0.0	10014	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10015	10	0.0	10015	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10016	10	0.0	10016	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10017	10	0.0	10017	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10018	10	0.0	10018	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10019	10	0.0	10019	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10020	10	0.0	10020	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10021	10	0.0	10021	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10022	10	0.0	10022	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10023	10	0.0	10023	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10024	10	0.0	10024	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10025	10	0.0	10025	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10026	10	0.0	10026	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10027	10	0.0	10027	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10028	10	0.0	10028	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10029	10	0.0	10029	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10030	10	0.0	10030	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10031	10	0.0	10031	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10032	10	0.0	10032	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10033	10	0.0	10033	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10034	10	0.0	10034	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10035	10	0.0	10035	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10036	10	0.0	10036	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10037	10	0.0	10037	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10038	10	0.0	10038	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10039	10	0.0	10039	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10040	10	0.0	10040	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10041	10	0.0	10041	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10042	10	0.0	10042	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10043	10	0.0	10043	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10044	10	0.0	10044	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10045	10	0.0	10045	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10046	10	0.0	10046	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10047	10	0.0	10047	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10048	10	0.0	10048	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10049	10	0.0	10049	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10050	10	0.0	10050	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10051	10	0.0	10051	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10052	10	0.0	10052	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10053	10	0.0	10053	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10054	10	0.0	10054	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10055	10	0.0	10055	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10056	10	0.0	10056	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10057	10	0.0	10057	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10058	10	0.0	10058	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10059	10	0.0	10059	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10060	10	0.0	10060	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10061	10	0.0	10061	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10062	10	0.0	10062	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10063	10	0.0	10063	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10064	10	0.0	10064	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10065	10	0.0	10065	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10066	10	0.0	10066	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10067	10	0.0	10067	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10068	10	0.0	10068	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10069	10	0.0	10069	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10070	10	0.0	10070	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10071	10	0.0	10071	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10072	10	0.0	10072	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10073	10	0.0	10073	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10074	10	0.0	10074	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10075	10	0.0	10075	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10076	10	0.0	10076	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10077	10	0.0	10077	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10078	10	0.0	10078	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10079	10	0.0	10079	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10080	10	0.0	10080	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10081	10	0.0	10081	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10082	10	0.0	10082	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10083	10	0.0	10083	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10084	10	0.0	10084	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10085	10	0.0	10085	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10086	10	0.0	10086	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10087	10	0.0	10087	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10088	10	0.0	10088	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10089	10	0.0	10089	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10090	10	0.0	10090	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10091	10	0.0	10091	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10092	10	0.0	10092	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10093	10	0.0	10093	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10094	10	0.0	10094	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10095	10	0.0	10095	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10096	10	0.0	10096	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10097	10	0.0	10097	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10098	10	0.0	10098	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10099	10	0.0	10099	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10100	10	0.0	10100	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10101	10	0.0	10101	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10102	10	0.0	10102	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10103	10	0.0	10103	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10104	10	0.0	10104	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10105	10	0.0	10105	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10106	10	0.0	10106	NEWMAN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10107	10	0.0	10107	SLAVIN	ANALYST
09-22-1980	10	NEW YORK	1000.0	10108	10	0.0	10108	PERCIVAL	ANALYST
09-22-1980	10	NEW YORK	1000.0	10109	10	0.0	10109	SMITH	CLERK
09-22-1980	10	NEW YORK	1000.0	10110	10	0.0	10110	JOHNS	MANAGER
09-22-1980	10	NEW YORK	1000.0	10111	10	0.0	10111	ADAMS	CLERK
09-22-1980	10	NEW YORK	1000.0	10112	10	0.0	10112	NEWMAN	ANALYST

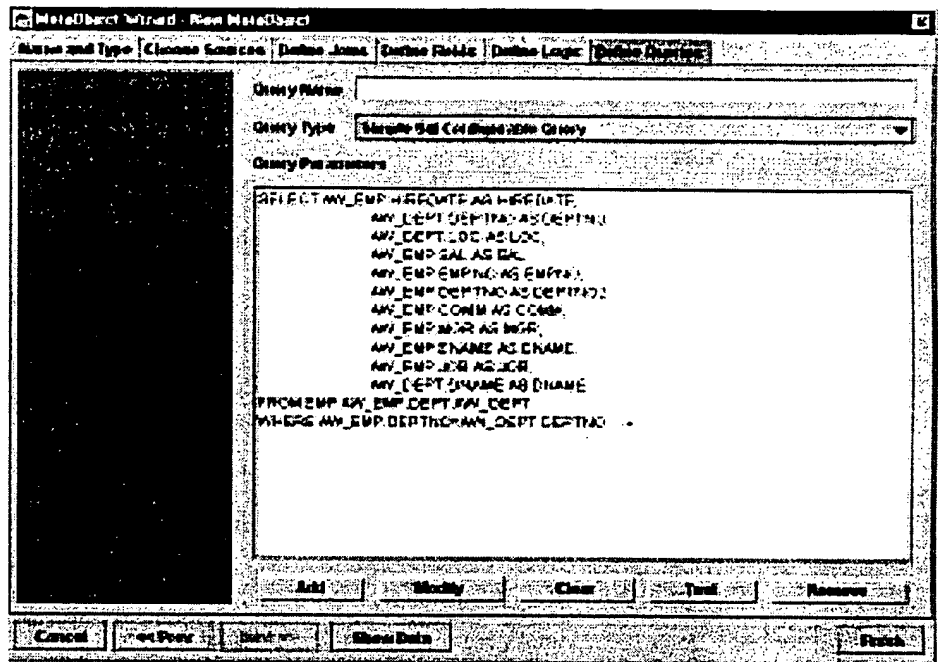
## Define Queries Tab

The Define Queries tab supports advanced features of AltoStudio and is used to either define a new query or to modify an existing query which is used by a MetaObject.

The use of this tab is recommended for the advanced user.



The purpose of the Define Queries tab is to allow Database Administrators to write simple, or advanced, SQL queries.



If, for example, you wanted to change all employee names and departments where the names and the departments are the same then use a simple query.

1. Identify the Query Name field as HELEN (it could be anything, of course).
2. Next, select Simple Sql Configurable Query from the drop-down list available in the Query Type field.

- [illegible]



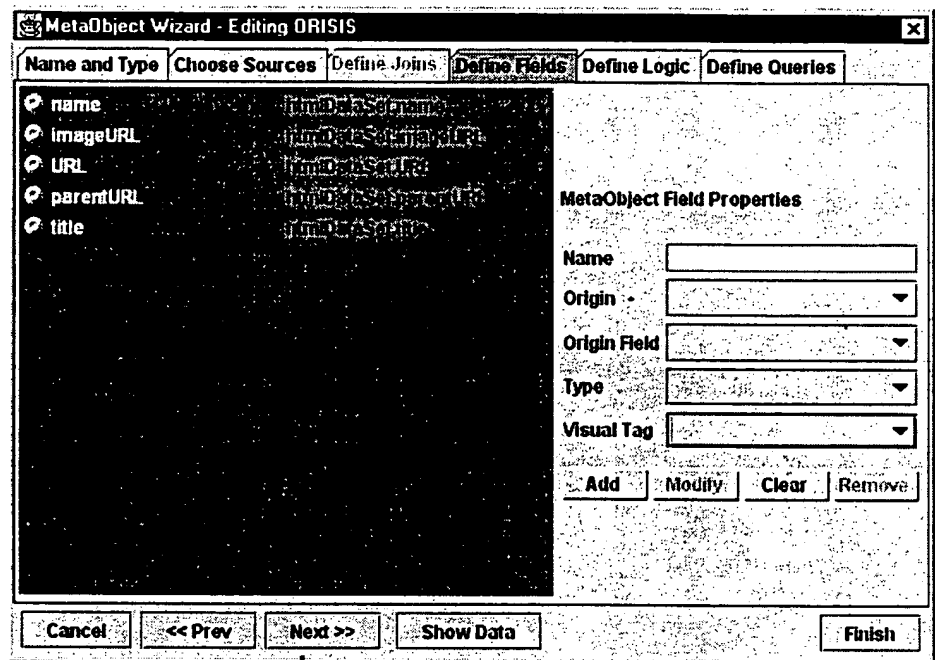
- 1

•

---

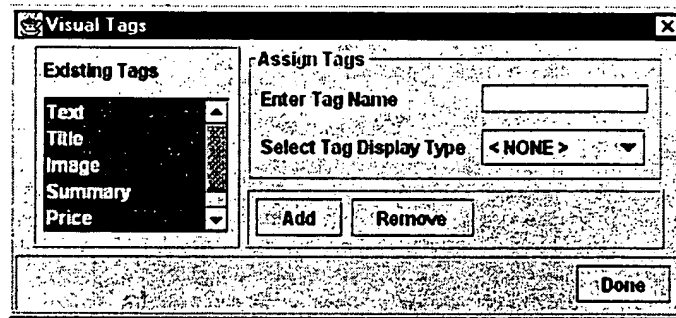
## Quick Access Add Visual Tabs Icon

This icon can be used to create alternative visual tabs for the **MetaObject Wizard > Define Fields > Visual Tag** field which is shown below in the Define Fields tab.

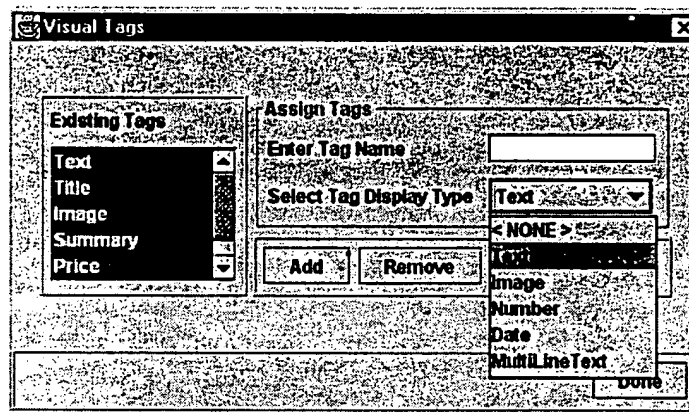


Click the Quick Access Add Visual Tabs icon .

The Visual Tags dialog box opens. All existing tags are shown in the Existing Tags dialog box.



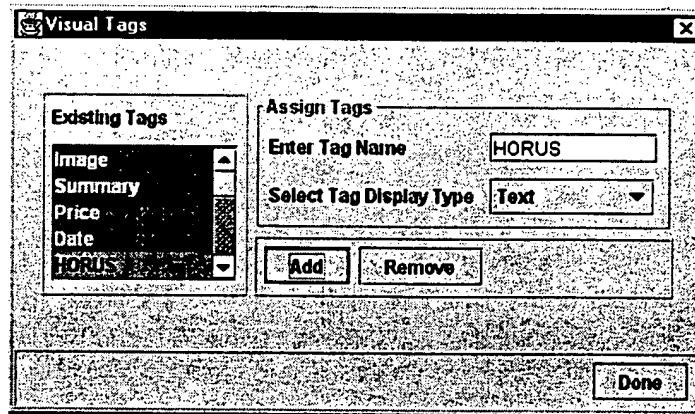
If we assume you would like to create a new visual tag that will be used to identify a field named next, place the cursor on the drop-down arrow and highlight the type of display you want.



In this example, we selected Text.

Since it would be convenient to name the tab, type HORUS into the **Assign Tags > Enter Tag Name** field.

The visual tag you have just created, HORUS, appears in the Existing Tags field and the tag display type is Text.

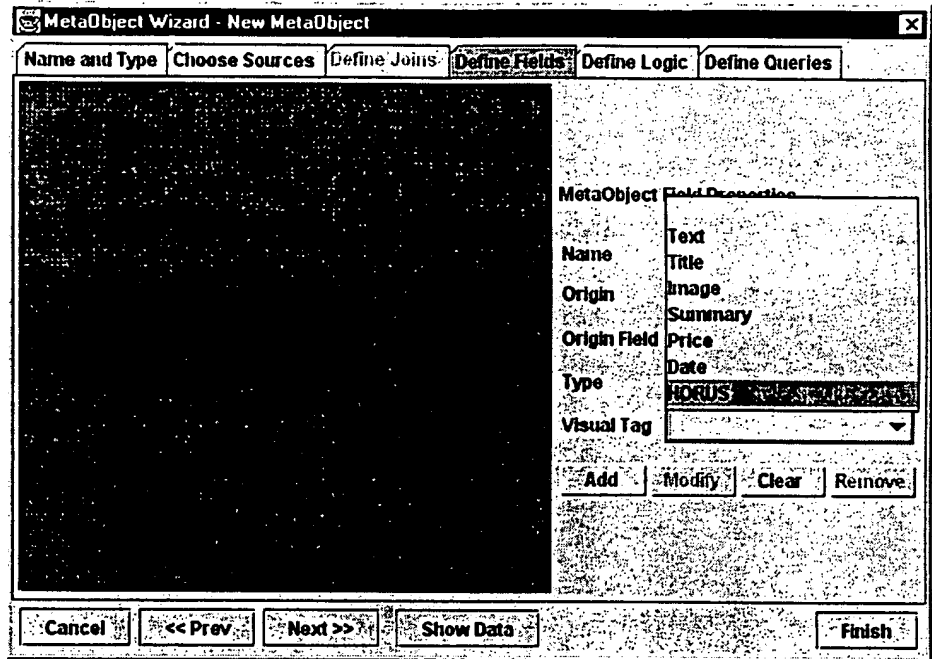


Click **Add**.

Note that HORUS is added to your list of Existing Tags.

Click **Done**.

The newly assigned, HORUS, visual tab is now available in the **MetaObject Wizard > Define Fields > Visual Tag** field (See "Define Fields tab" on page 83.) as a Visual Tag.

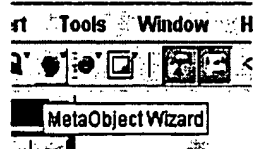




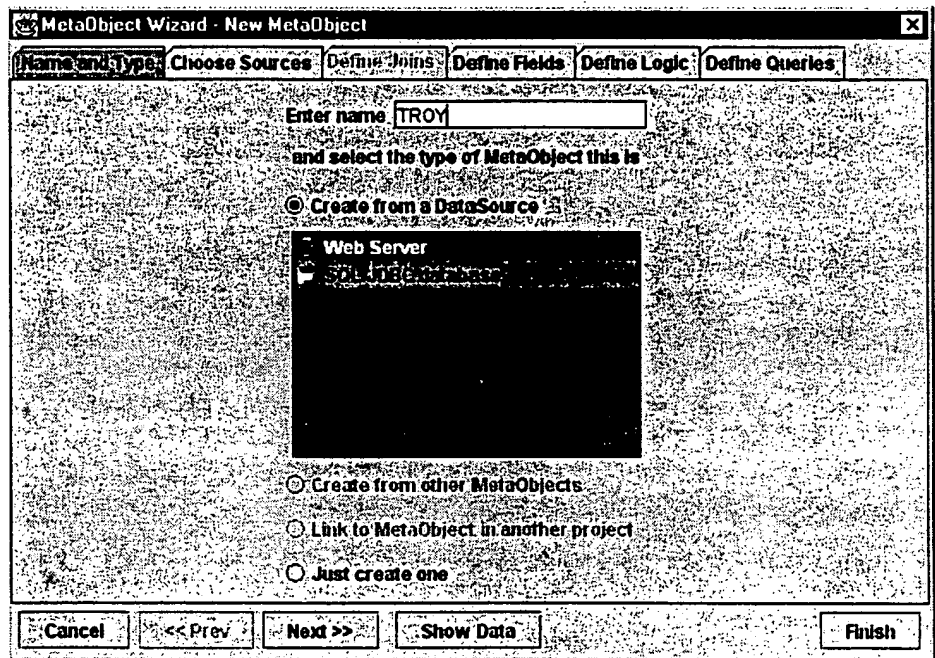
---

## MetaObject Wizard Icon

An alternative method of creating MetaObjects is to use the Quick Access tool bar MetaObject Wizard icon.



Clicking the MetaObject Wizard icon will activate the MetaObject Wizard window.

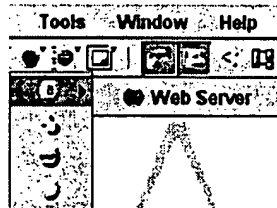


If you click and hold your cursor on the MetaObject Wizard icon you will see a drop-down list of available types of MetaObjects.

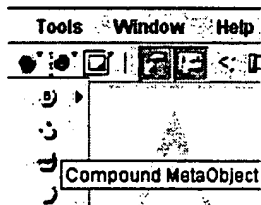
The four drop-down alternatives from the MetaObject Wizard icon are:

- **DataSet MetaObject.** A DataSet MetaObject is defined as a MetaObject that is associated with a single database. A DataSet MetaObject comes directly from the server.

In the example shown below the database that you previously selected in the DataSource Wizard is Web Server. The data sets that are exported from a Web Server are URL's, whereas the data sets that are exported from a SQL/JDBC database are known as tables in the MetaObject Wizard tabs.

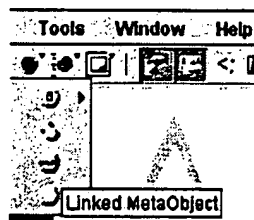


- **Compound MetaObject.** A Compound MetaObject is a MetaObject that is derived from other MetaObjects that have been joined.



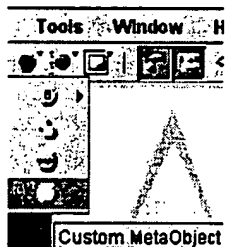
- **Linked MetaObject.** A linked MetaObject is a MetaObject that has been previously created in another project and is going to be used in this project as well.

Note that, while you cannot modify a MetaObject you have imported from another project, you can read it to your existing MetaObject.



- **Custom MetaObject.** A Custom MetaObject is one in which you, using business logic, determine the contents of the fields. A Custom MetaObject can be independent of a data source when created then, subsequently, linked to a data source.

A Custom MetaObject consists of fields that do not come from a given source but originate in the **MetaObject Wizard > Define Fields** tab. Use the **Define Fields > Add** button to create a MetaObject without linking to a specific data source by not placing data in the Origin, Origin Field, and Type fields.



For example, let us assume your organization has purchased a new DB2 database from IBM but it will not arrive for another two weeks. You can use the Custom MetaObjects area to create a new MetaObject. Then, when the new DB2 database is installed and operational, all you have to do is add the field values.

In short, the Custom MetaObject field allows you to create a "skeleton" MetaObject which can be implemented later.

You are now ready to map the MetaObjects you have created.

Chapter 7, "Connecting MetaObjects" on page 101, shows you how to integrate MetaObjects into a usable configuration.

---

At this point, you have created a project, PROMETHEUS, linked it to a server identified as t3://qa-oracle1:7001, and associated it with a Web Server Data Source. You have also created MetaObjects which will be used to populate your project.

This chapter will show you how to connect MetaObjects to each other. These links, in turn, produce interrelationships that are used in the AltoClient.

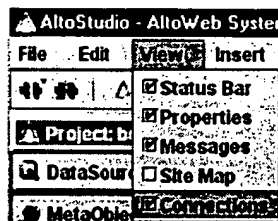
This chapter includes the following topics:


- "Accessing Connections" on page 102
- "Relationship Diagram" on page 102
- "Connection Wizard" on page 105

---

## Accessing Connections

From the AltoStudio window menu bar, access **View > Connections** by placing a check mark in the box and clicking your mouse. This will open the Relationship Diagram window.



Note: An Alternative method would be to use the Quick Access tool bar  Show Connections icon.

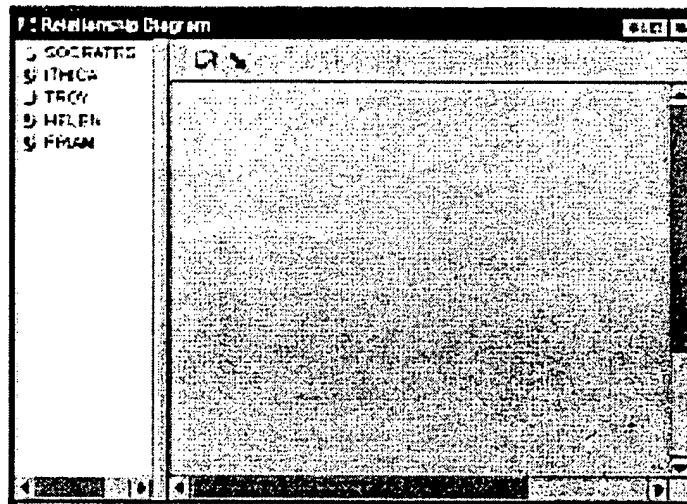
---

## Relationship Diagram

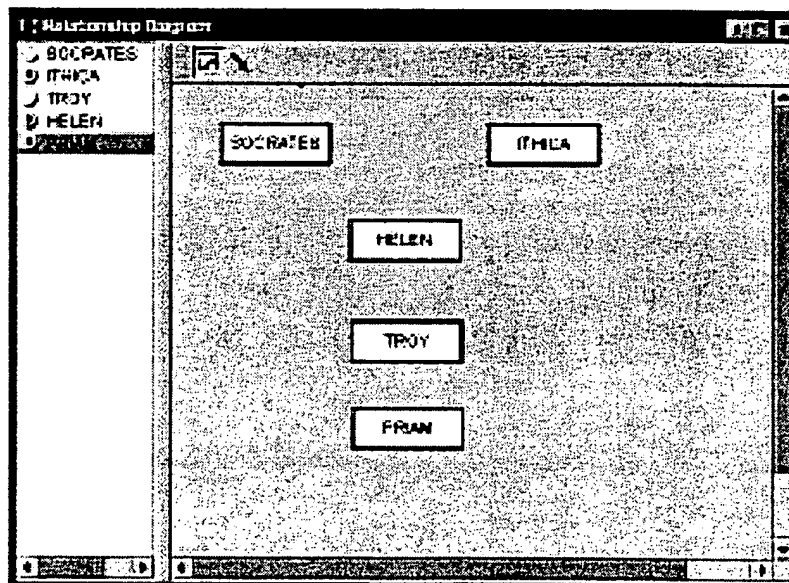
The Relationship Diagram window takes the MetaObjects you have created in your project, or projects, and allows you to interconnect them.

Either clicking the Quick Access Show Connections icon or using the menu bar, **View > Connections** will open a Relationship Diagram window.

A Relationship Diagram window with five MetaObjects is shown below.

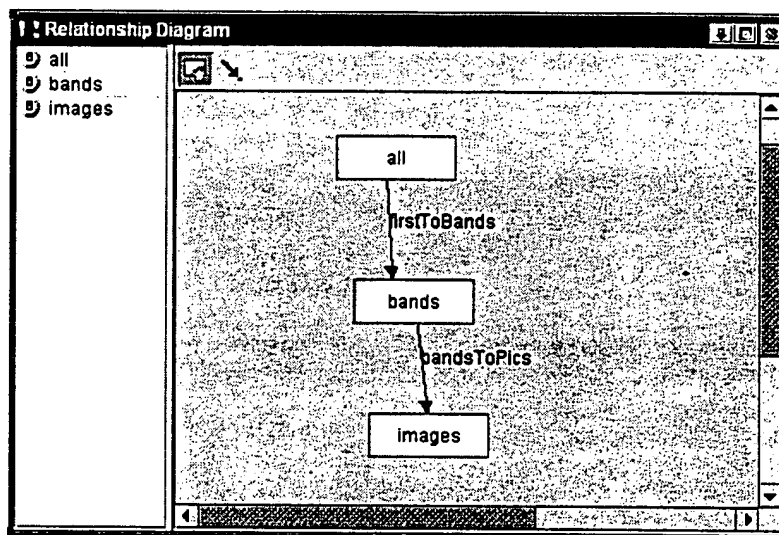


Highlight any MetaObject. Place your cursor in the open field and, holding the button, draw your MetaObject box.



The placement of a MetaObject is not important since it can be moved, and reshaped, at any time.

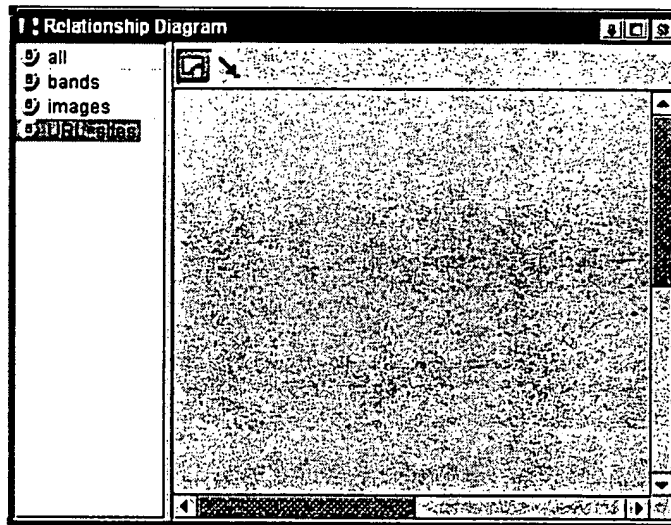
In this more advanced illustration, the MetaObjects ALL, BANDS and IMAGES have been placed in the window, connected, and the connections named using the Connection Wizard which is discussed next.



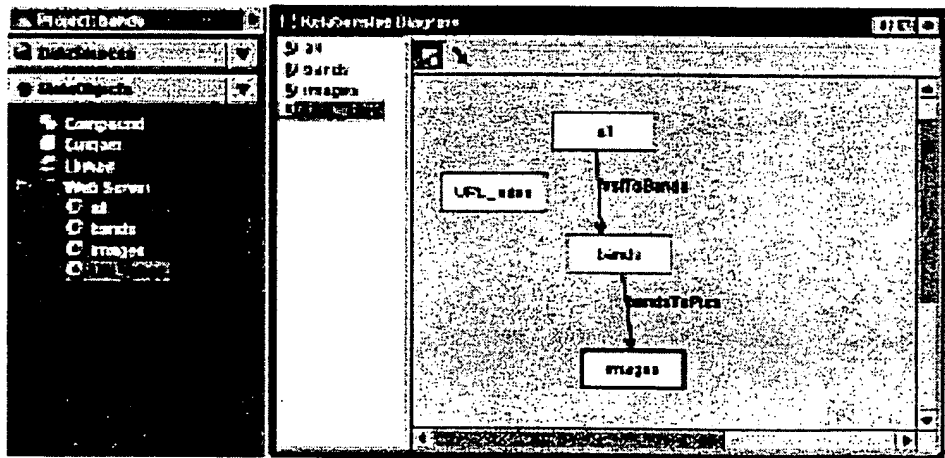


## Connection Wizard

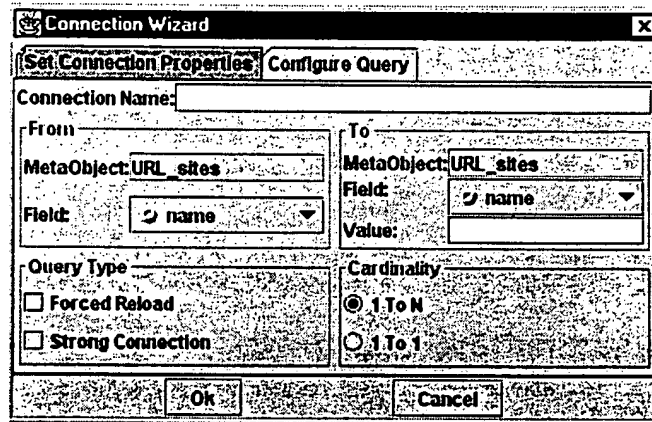
Highlighting and clicking any MetaObject in the Relationship Diagram window will automatically open the Connection Wizard dialog box.



In this example, we have added a new MetaObject, URL\_sites and placed it in the Relationship Diagram window.



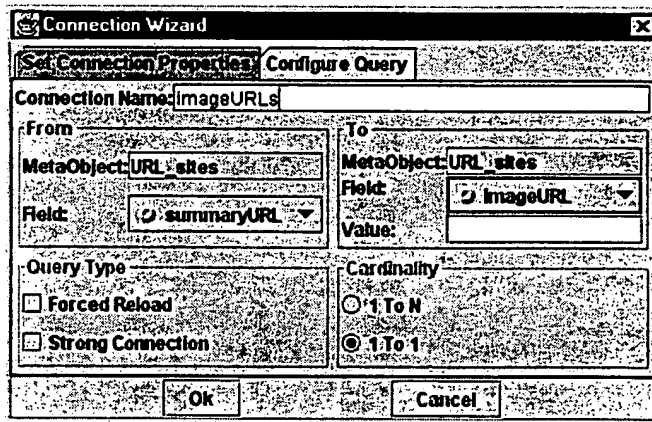
Highlighting and clicking URL\_sites will open the Connection Wizard dialog box.



### Set Connection Properties tab

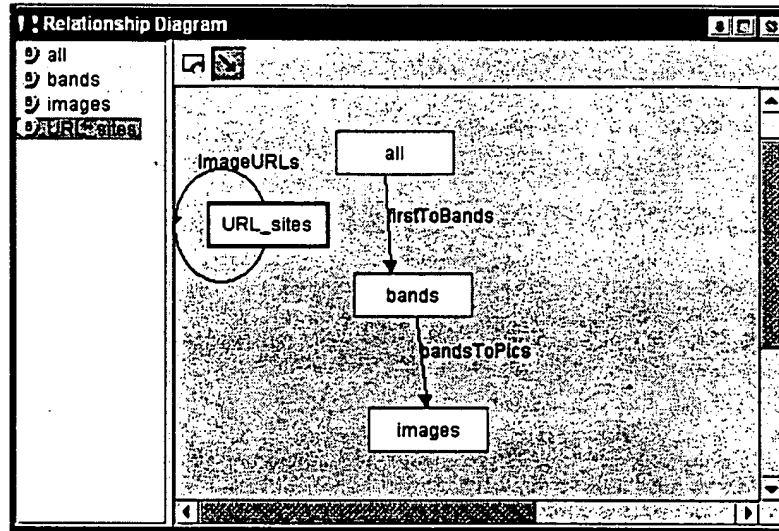
The Set Connection Properties tab is used to define and name the relationships between MetaObjects. In this example, we have defined a circular loop. We can also connect it to any other MetaObject if we want.

We must now assign a name to the connection (image URLs).



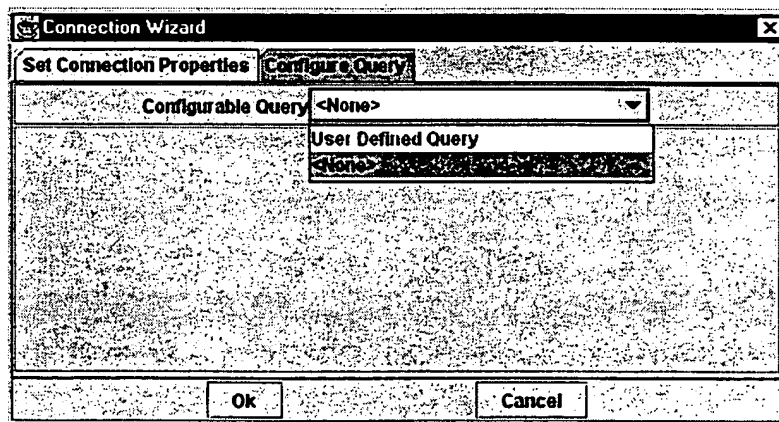
Click **OK**.

The result of these connections and identities is shown below.



## Configure Query tab

The Configure Query tab is used to define queries within relationships.



Click **OK**.

You are now ready to create the visual aspects of the representation of your project using Chapters 8 and 9.

Copyright © 2008

---

So far, you have created a project, Prometheus, linked it to a server identified as `t3://qa-oracle1:7001`, and associated it with a Data Source.

You have assigned a new URL to your project, `www.yahoo.com`, and populated it with a MetaObject. You have placed URL information into that MetaObject and reidentified the MetaObject.

You are now ready to incorporate this data into a template and design the appearance of your screen.

This chapter includes the following topics:

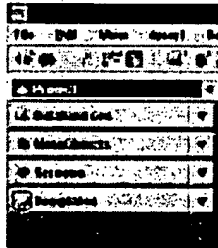
- "To View or Modify an Existing Template" on page 110
- "Creating a Template" on page 111
- "Insert > New Template" on page 111
- "Quick Access Template Editors Icon" on page 112
- "Insert > New Screen" on page 125
- "Quick Access Screen Editors Icon" on page 125

---

## *To View or Modify an Existing Template*

---

It is possible to view all existing templates as well as edit their properties. From the menu bar access **File > New Project > Project > Templates**.



Click the icon within the field named Templates. It will move up to just below the Screens field.

Clicking the drop-down arrow in the right side of the Template field brings up a dialog box with two choices.


- View by Name
- View by Type

All existing templates are listed below the Template field. Clicking any of these fields will open the Templates Editor window for that type of template as well as show the existing properties of that template.

---

## *Creating a Template*

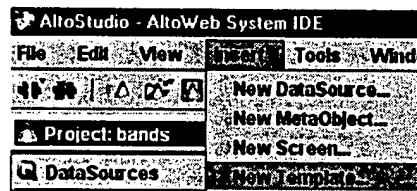
There are two ways to create a new, or modify an existing, template:

- From the menu bar: **Insert > New Template**
- From the Quick Access tool bar **Template Editors** icon 

---

## *Insert > New Template*

From the AltoStudio window menu bar, access **Insert > New Template**.




Clicking this field will open the Template Editors. As the functionality of this process is covered by the Quick Access Template Editors icon, it will not be duplicated here.

---

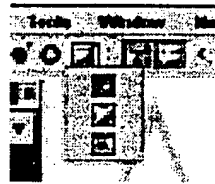
## *Quick Access Template Editors Icon*

---

An alternative method to using the menu would be to use the Quick Access Icon Template Editors  button.



If you click and hold your cursor on the Template Editors icon you will see a drop-down list of available types of editors.



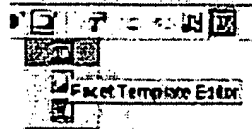
The three drop-down alternatives from the Quick Access Template Editors icon are, from top to bottom:

- Facet Editor
- Context Editor
- Structure Editor

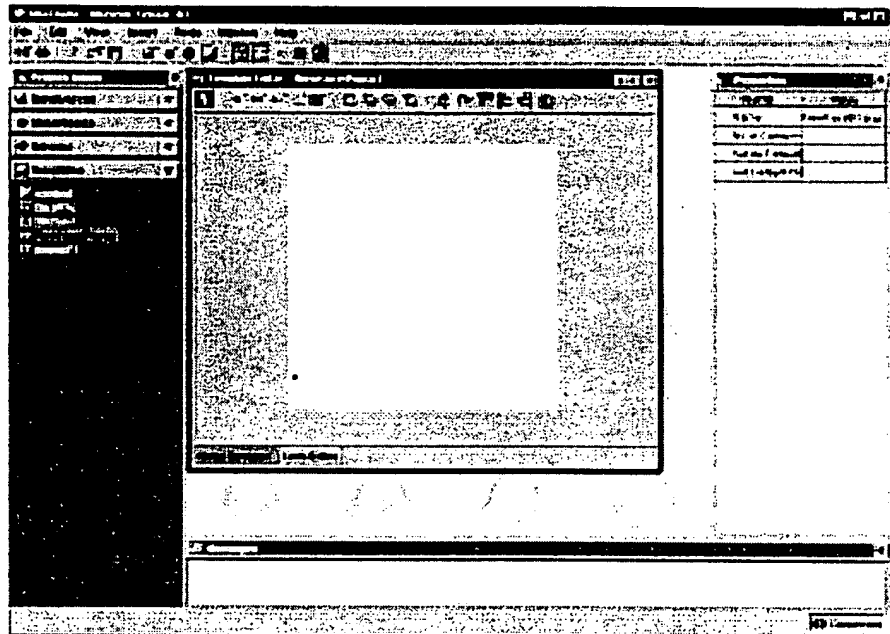


## Template Editor — NewFacetPopup

Hold and drop your cursor down the Template Editors icon until the Facet Template Editor is highlighted.

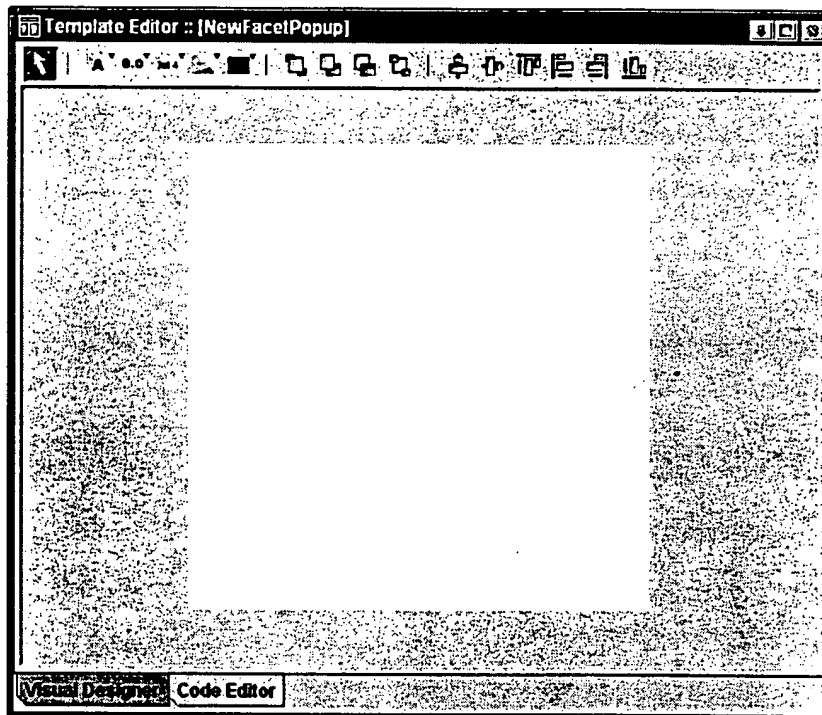


This will open the Template Editor:[NewFacetPopup] window. This window is used to determine the appearance and content of your facet pop-ups. This information could be either text or image.



Two tabs are located at the bottom of the screen:

- Visual Designer
- Code Editor



### ***Visual Designer***

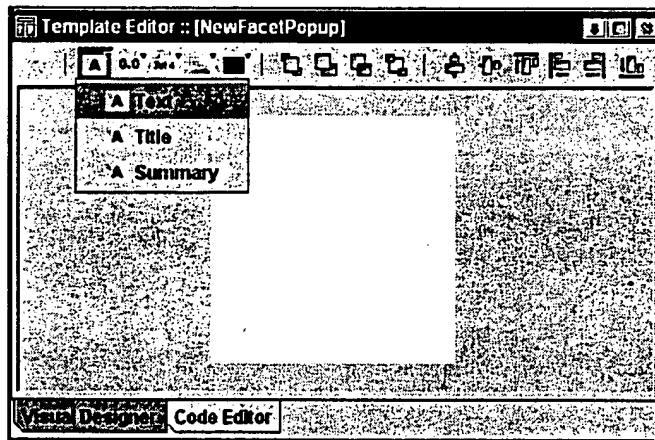
Accessing the Visual Designer tab allows you to insert text, dates, numbers, and images into your facet popup. These are represented by the first five icons reading from left to right.



Each of these first five icons have drop-down arrows.

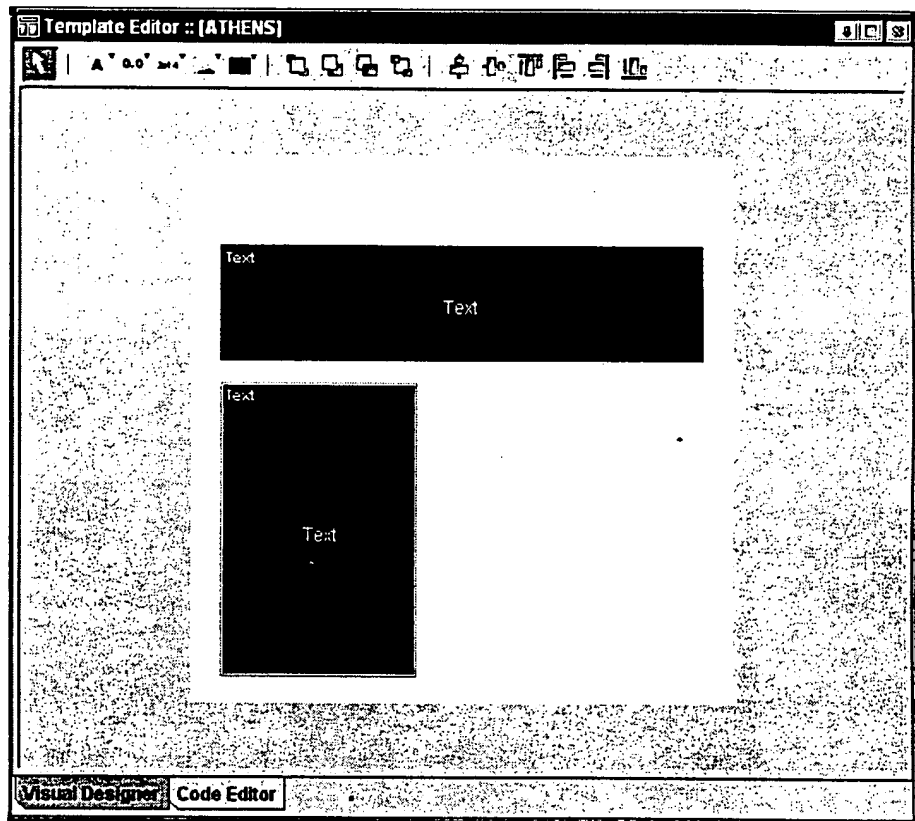
The alternatives are available by clicking the arrow and selecting one of the drop-down selections.

The illustration below shows the text icon with three drop-down alternatives: Text, Title, Summary. In this example, Text is selected.



Place the cursor cross-hairs into the window and, holding down the mouse button, draw a text box in the window. The shape and position of this box is immaterial since it can be reshaped and moved at any time.

A typical template editor window with multiple text boxes might resemble the following.



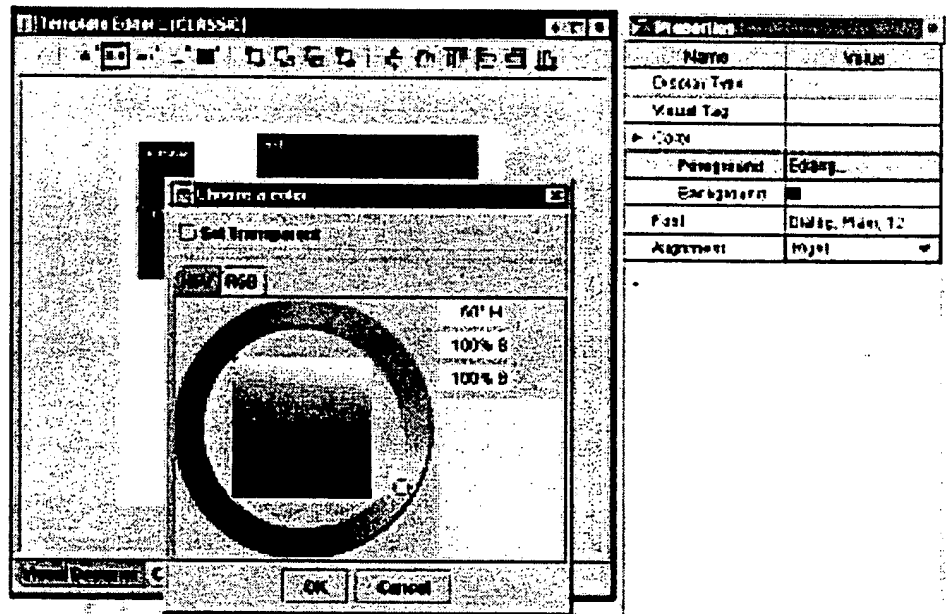
Once you have created a text box you can design and modify — using the Properties dialog box — the textual information that will be presented in the facet popup.

That is, highlight the field, click backspace or delete in the Properties dialog box, and type your information.

Clicking **Enter** or **Return**.

You can also use the Template Editor Properties dialog box to change both the foreground and background colours of your text.

In this example, double-click the Foreground Color in the Properties dialog box. The Value changes to Editing... and the **Choose A Color** window appears.



Choose your colour and click **OK**.

All changes will be reflected in your Visual Designer text box.

An image box, as compared to a text box, is a place-holder for either an imported image from an acceptable source in your database or an image that is retrieved from a URL site. Once created, that URL image will point to that URL site.

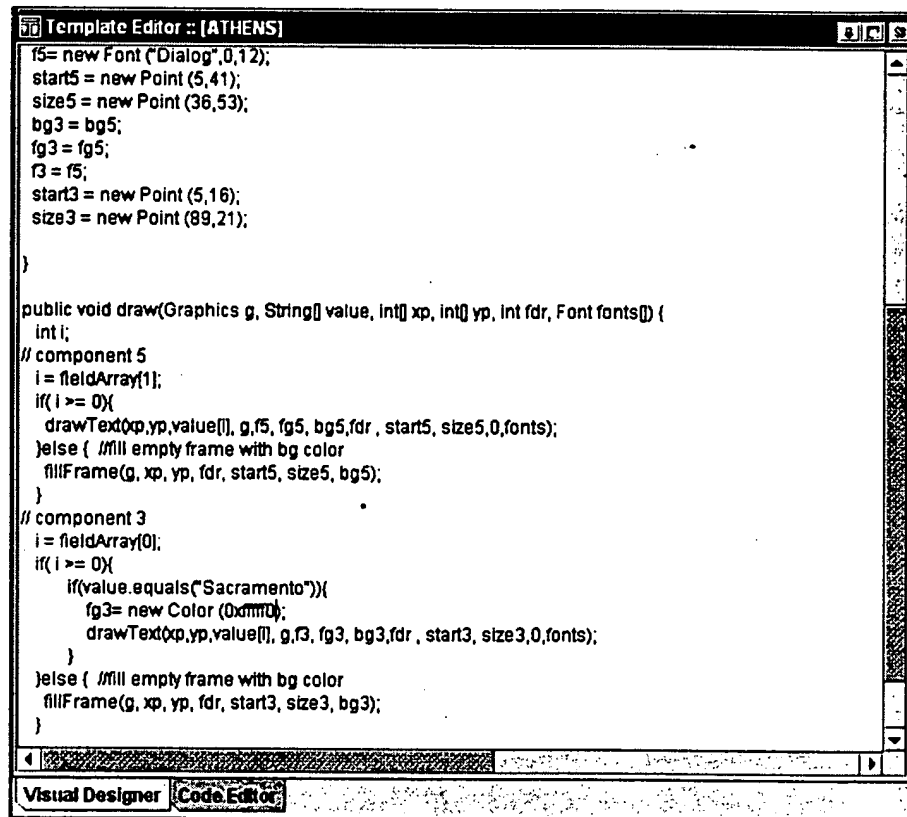
The other ten icons in the menu bar align or position your information within the frame. They are self-explanatory.

## Code Editor

Accessing the Code Editor tab allows you to edit the Java code that is used to create the user interface facet.

The Code Editor pane shows the software contents, in Java code, of the properties in the Template Editor:[NewFacetPopup] window, as well as the defaults for the facet popup.

A typical example of Java software code for the Template Editor is shown below.



```

Template Editor :: [ATHENS]

f5= new Font ("Dialog",0,12);
start5 = new Point (5,41);
size5 = new Point (36,53);
bg3 = bg5;
fg3 = fg5;
f3 = f5;
start3 = new Point (5,16);
size3 = new Point (89,21);

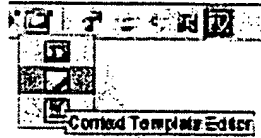
}

public void draw(Graphics g, String[] value, int[] xp, int[] yp, int fdr, Font fonts[]) {
    int i;
    // component 5
    i = fieldArray[1];
    if( i >= 0){
        drawText(xp,yp,value[i], g,f5, fg5, bg5,fdr , start5, size5,0,fonts);
    }else { //fill empty frame with bg color
        fillFrame(g, xp, yp, fdr, start5, size5, bg5);
    }
    // component 3
    i = fieldArray[0];
    if( i >= 0){
        if(value.equals("Sacramento")){
            fg3= new Color (0xffff00);
            drawText(xp,yp,value[i], g,f3, fg3, bg3,fdr , start3, size3,0,fonts);
        }
    }else { //fill empty frame with bg color
        fillFrame(g, xp, yp, fdr, start3, size3, bg3);
    }
}
    
```

The ability to change the software code presents both an opportunity to customize your presentation at the highest level and a potential source of incredible problems if you are not a Java software developer. We encourage caution.

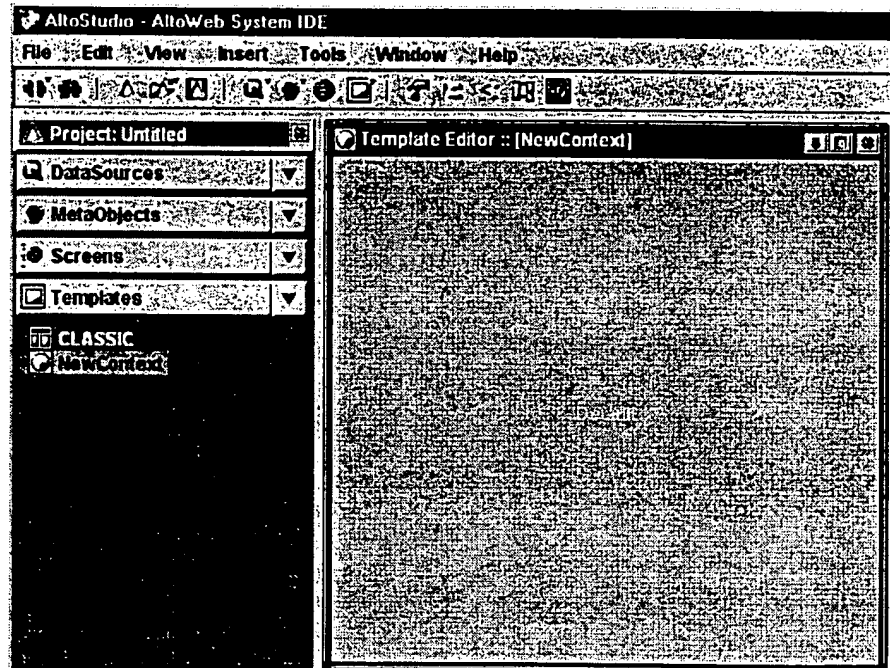
## Template Editor — New Context

Hold and drop your cursor down the Template Editors icon until the Context Template Editor icon is highlighted. Click this icon.



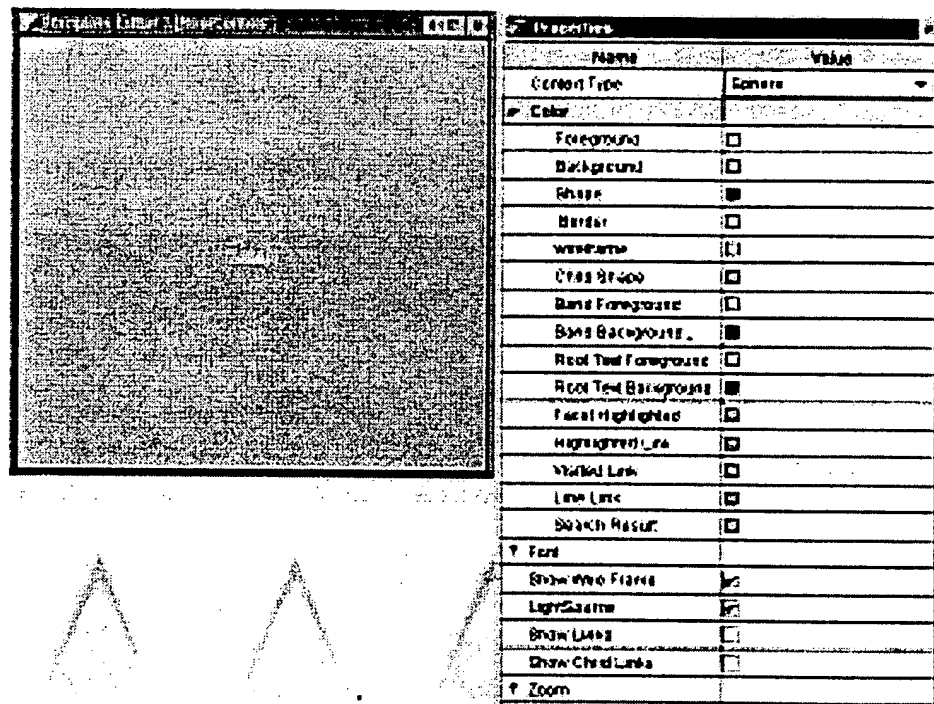
This will open the Template Editor:[NewContext] window. This window is used to determine the appearance of your context pane. The context pane contains the sphere or cube.

Highlighting and clicking the Context Template Editor icon will open the Template Editor::[New Context] and place an unnamed context icon in the Project/Templates pane.



Open the Properties dialog box for the Template Editor.

The fields within the Properties dialog box allow you to determine the shape of the image within the pane, whether a cube or sphere, as well as fonts, colors, links, and more.



For example, the determination of colours is determined by clicking any of the boxes in a colour field.



[illegible]

Click **OK**.

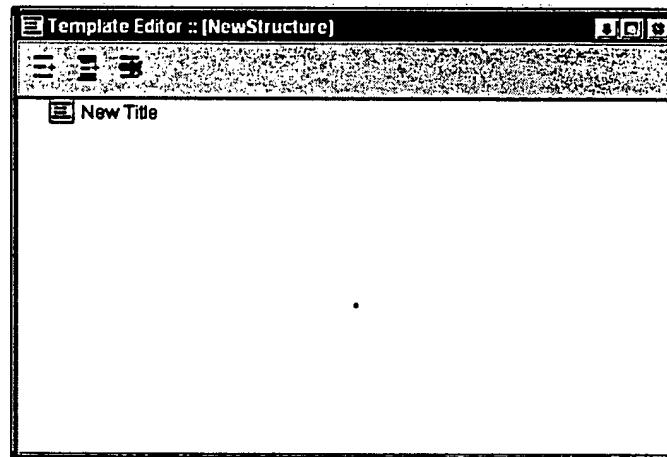
# AltoStudio

## Template Editor — NewStructure

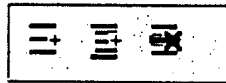
Hold and drop your cursor down the Template Editors icon until the Structure Template Editor icon is highlighted. Click this icon.



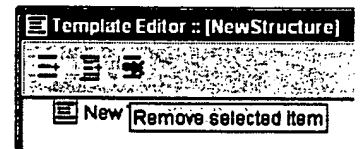
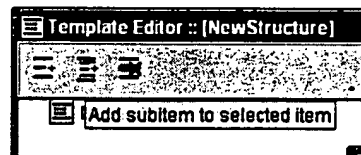
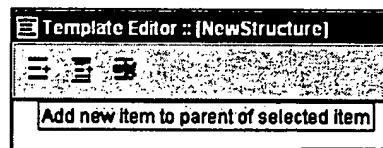
The basic function of Structure Editor:[NewStructure] is to determine the content of the frame of the AltoClient. A hierarchical field refers you to the stages in the Site Map you will set in the State Diagram (Chapter 9).



There are three icons in the menu bar.



Reading from left to right they are:



Use the Properties dialog box to identify the structure as well as modify and add information. For example, this template will be identified as ODYSSEUS and the URL link to [www.reel.com](http://www.reel.com) will be added to the Mouse Over/Content. This means that, when the mouse is placed over the facet identified as ODYSSEUS, the link to the URL [www.reel.com](http://www.reel.com) will be activated.

Properties	
Name	Value
Title	odysseus
Children	Static
Mouse Over	
Context	<None>
Content	<a href="http://www.reel.com">http://www.reel.com</a>
Mouse Click	

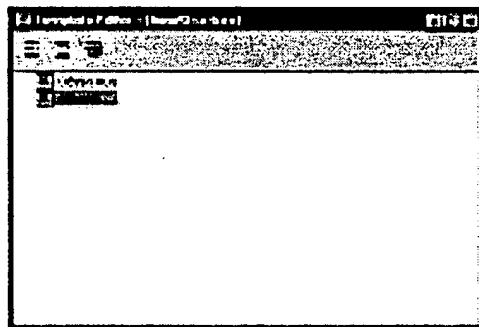
Additionally. You can designate a context for Stage 1 as a mouse click by using the drop down arrow and highlighting Stage 1.

Properties	
Name	Value
Title	odysseus
Children	Static
► Mouse Over	
Context	<None>
Content	http://www.reel.com
► Mouse Click	
Context	<None>
Content	<None>
	Stage 2
	Stage 3

Note: A stage is defined as a screen with commands to a location on the Site Map (See Chapter 9: "Site Maps" on page 135.).

This process can be expanded until you have populated your sphere to your particular requirements.

That is, use the Properties dialog box to identify a second structure to modify and add an additional facet. This template will be identified as ATHENA and the URL link to [www.amazon.com](http://www.amazon.com) will be added to the content when the mouse is clicked over the structure.



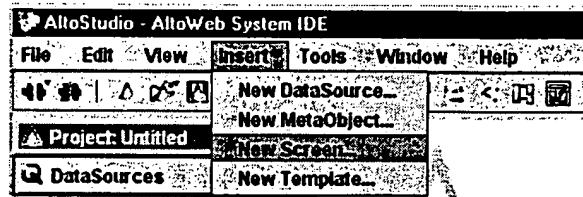
Properties	
Name	Value
Title	ATHENA
Children	Static
► Mouse Over	
Context	<None>
Content	
► Mouse Click	
Context	<None>
Content	http://www.amazon.com

You are ready to proceed to the development of your screen.

---

## *Insert > New Screen*


From the AltoStudio window menu bar, access **Insert > New Screen**.



Clicking this field will open the Screen Editor. As the functionality of this process is covered by the Quick Access Screen Editors icon, it will not be duplicated here.

---

## *Quick Access Screen Editors Icon*

Click and hold your cursor on the Screen Editors icon .

This will open two superimposed screens:

- Context Screen .Properties
- Context Editor

### **Context Screen Properties**

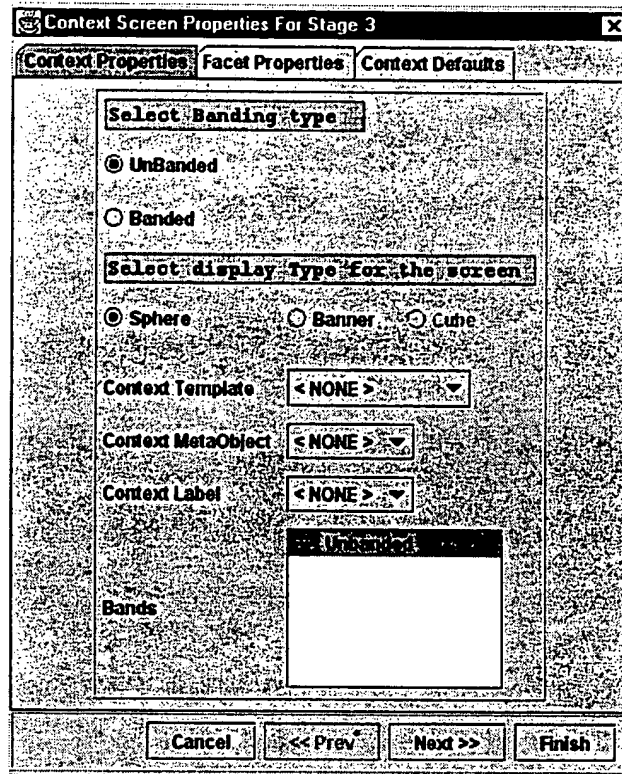
Click the Context Screen Editors icon to access the Context Screen Properties for Stage (n).

Note that there are three tabs:

- Context Properties
- Facet Properties
- Context Defaults

## Context Properties

The Context Properties window is used to determine the type of banding as well as the type of screen display.



The Context Screen Properties for Stage 3 is shown above.

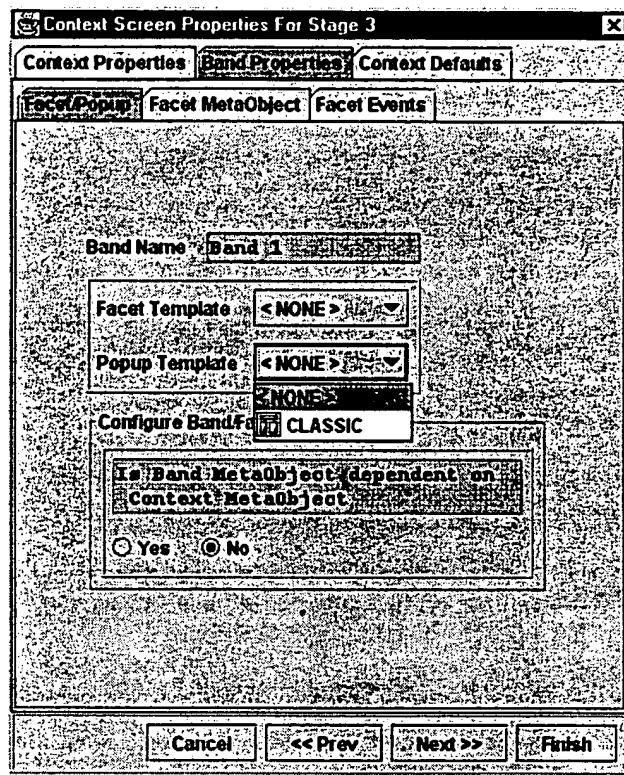
The term Unbanded means that you are pulling information from only one MetaObject. Banded means you are pulling information from two, or more, MetaObjects. Each field supports drop-down arrows that contain the information you have previously developed.

Press **Next**.

### Facet (Band) Properties

If your sphere is unbanded, you will see the Facet Properties dialog box.

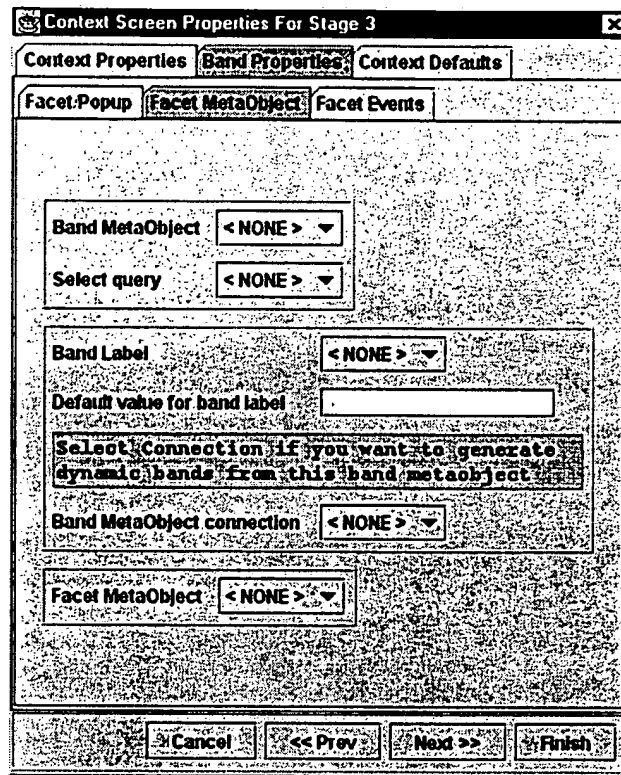
If you choose banded, then you will see the Band Properties dialog box.



The three tabs in this dialog box allow you to name and determine the visual characteristics of the facets within your band. Use the several fields and drop-down features to populate the band.

Press **Next**.

The Facet MetaObject tab opens.



This dialog box is used to label and associate your band with MetaObject connections.

Band Label represents a dynamic configuration environment. This means, in turn, that you can have primary and secondary subsets in the band.

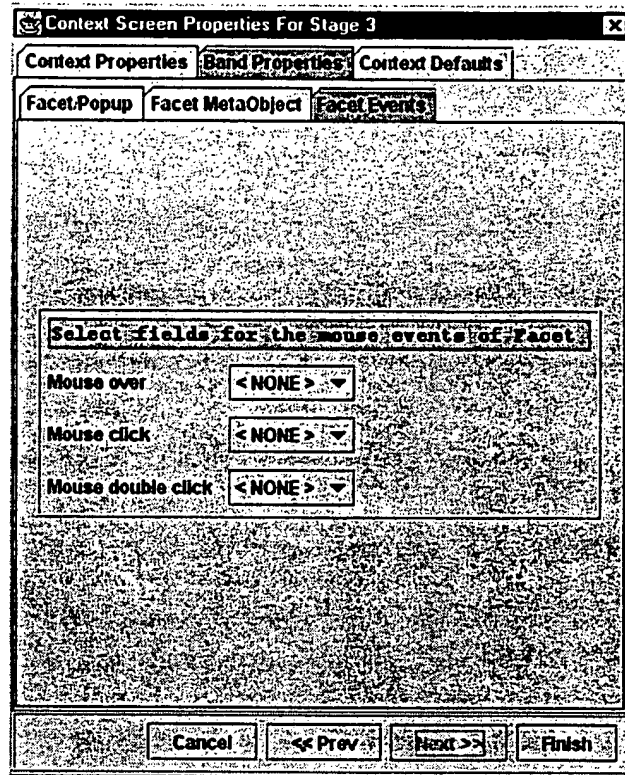
For example, a dynamic configuration might consist of the primary generic set termed CD's which could support secondary subsets such as Classical CD's, Western CD's, and more.

Use the other several fields and drop-down features to populate the band.



Press **Next**.

The Facet Events tab opens.



This tab allows you to determine the type and characteristics of your mouse to the facet. Use the drop-down arrows to facilitate your selections.

Click **Finish**.

## Context Defaults

The Context Defaults tab is used to determine the source of stage default values.

Context Screen Properties For Stage 3

Context Properties Band Properties Context Defaults

Where do you want to select default value for the stage from?

☒ Field ☐ Query

Select default field < NONE >

Enter value for field

Cancel << Prev Next >> Finish

Use the drop-down arrows to facilitate your selections.

Click **Finish**.

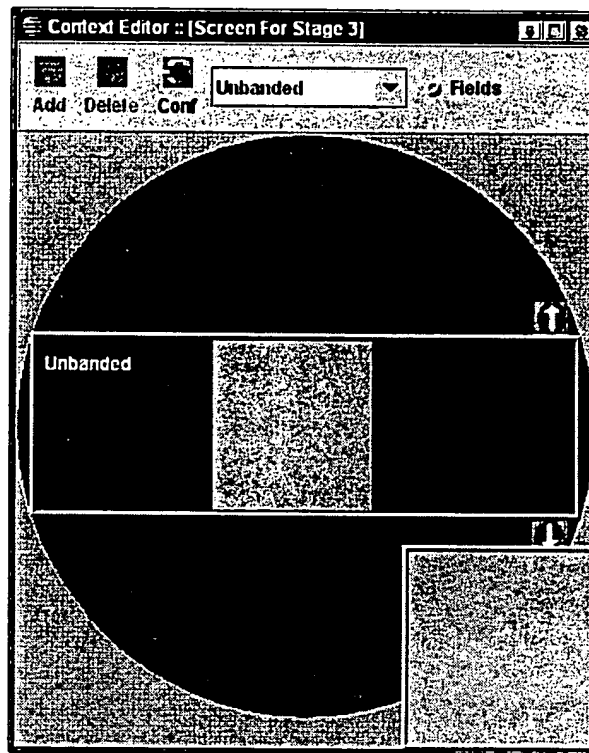
---

## Context Editor

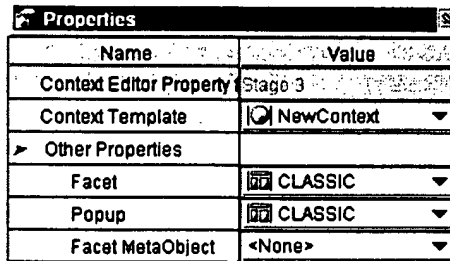
---

The Context Editor allows you to add, or delete, bands and fields as well as place information within the bands.

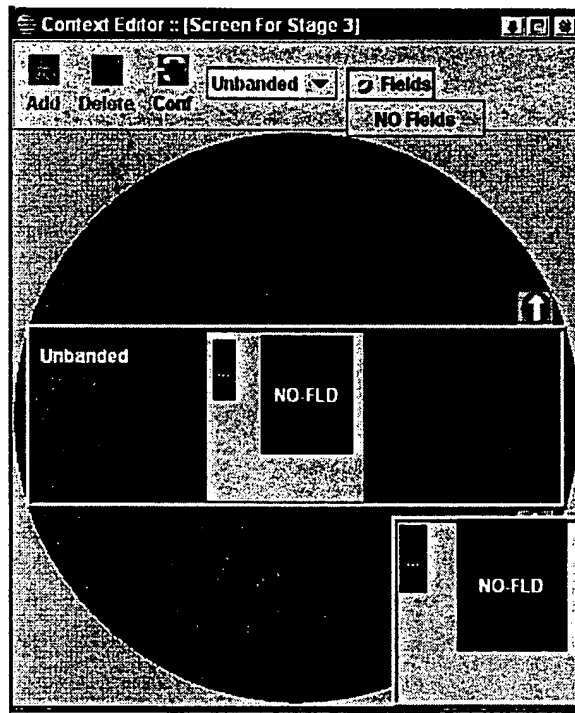
Multiple clicks to the icon will produce subsequent screens.



The first step in configuring your Context Editor is to open the Properties dialog box.



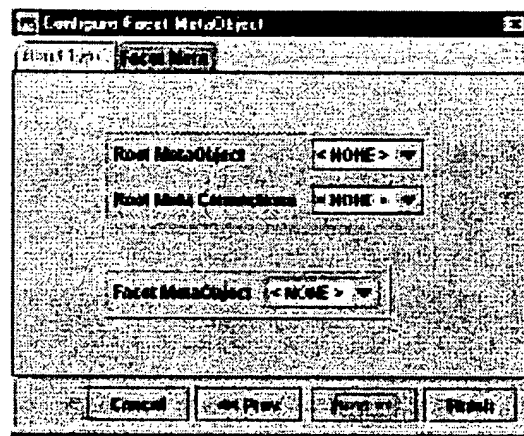
Use the Properties dialog box to determine the characteristics of your band. The drop-down arrows contain the information you have previously developed. For example, CLASSIC, is selected for the facet and popup as shown below.



Note that the up arrow is used to access the previous band while the down arrow can be used to access the subsequent band.

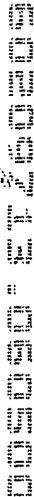
Click **Next**.

The next tab in the Configure Facet MetaObject window is Facet Meta.



A Root MetaObject a hierarchical system. For example, it might be identified by, and could consist of, a Department. Given this, a Root Meta Connection might be identified as all Employees in the Department that are identified with the Facet MetaObject named Employees.

Press **Finish**.



1

---

So far, you have created a project, Prometheus, linked it to a server identified as `t3://qa-oracle1:7001`, and associated it with a Data Source.

You may also have associated it with a new URL, `www.amazon.com.`, populated it with a MetaObject and identified that MetaObject. You have placed URL information into that MetaObject and incorporated this data into a template.

You are now ready to integrate this information into a site map. A site map defines how screens are connected together. It defines whether connections are used or not used.

This chapter includes the following topics:

- "Creating a Site Map" on page 136
- "Site Map Window" on page 138


---

## *Creating a Site Map*

---

A Site Map is used to connect stages together. Recall from Chapter 8 that a stage consists of a screen and, in an optimum environment, a structure with commands to a location on the Site Map.

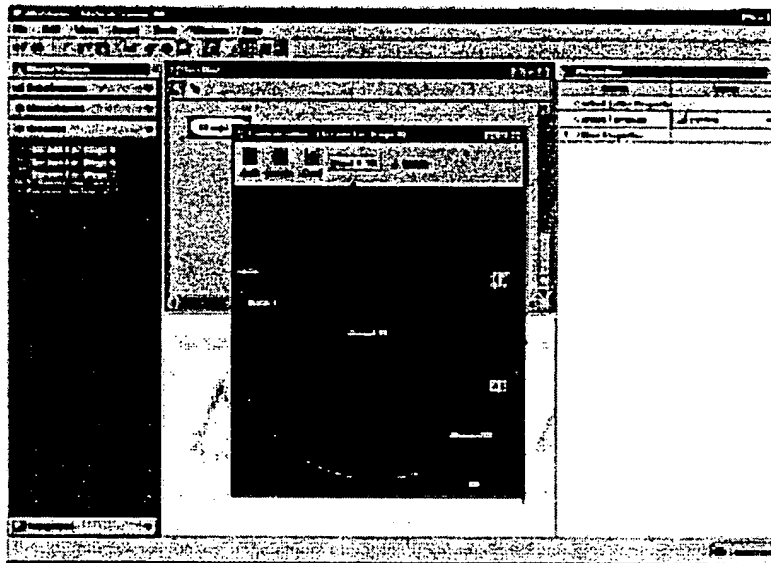
There are two ways to access Site Map functionality:

- From the menu bar: **View > Site Map**
- From the Quick Access tool bar Show Site Map  icon

The purpose of creating a site map is to establish routes between stages. That is, to determine how stages are connected together.

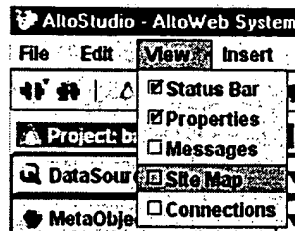
In fact, the basic function of a site map is that of a visual correspondent to the Relationship Diagram.

A typical site map with an associated Context Editor might resemble the following illustration.

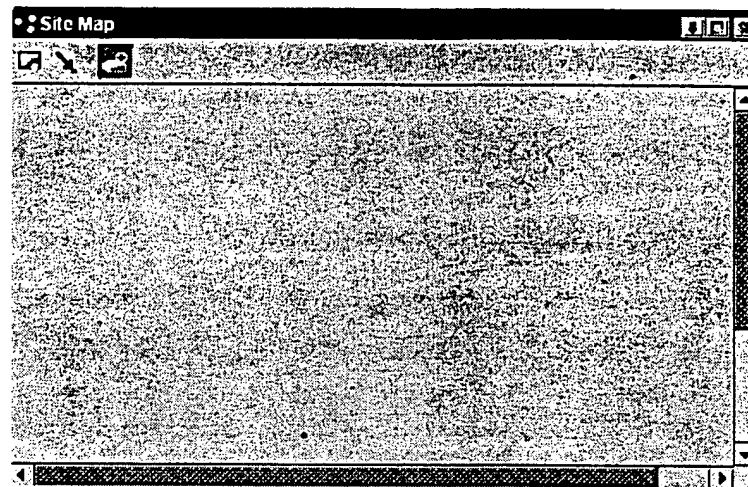




From the AltoStudio window menu bar, access **View > Site Map**.



This will access the Site Map window.



Note: You can also click the Quick Access tool bar Show Site Map icon to access the Site Map window.



---

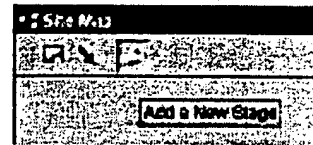
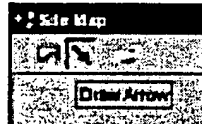
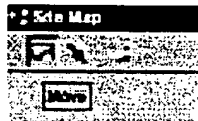
## Site Map Window

---

The three icons in the Site Map window menu bar are shown below.

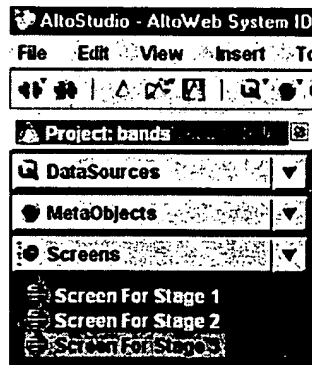
From left to right:

- The Move icon allows you to move any stage in the window
- The Draw Arrow icon allows you to connect stages
- The Add a New Stage does just that

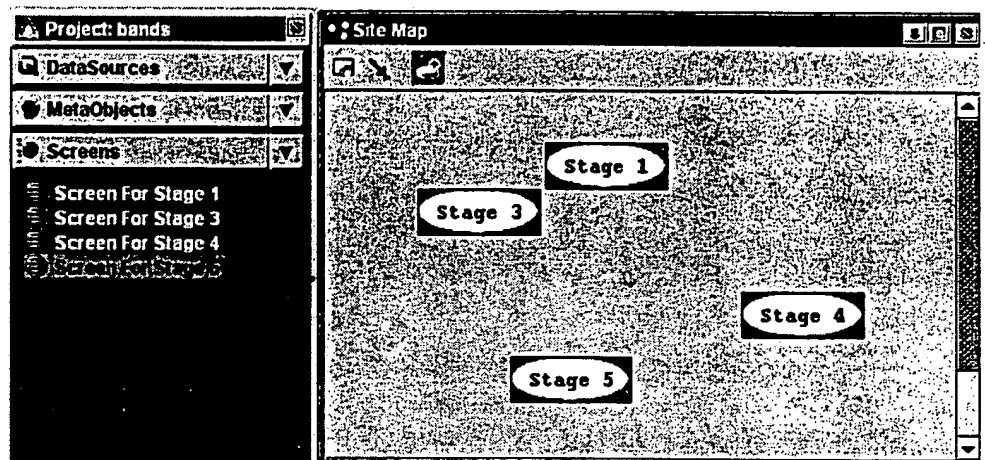


A site map is populated with the stages you have created in your Project using the Context Editor and the Quick Access Screens icon or the Add a New Stage icon in the Site Map.

Stages will be generically listed in the Project under Screens.

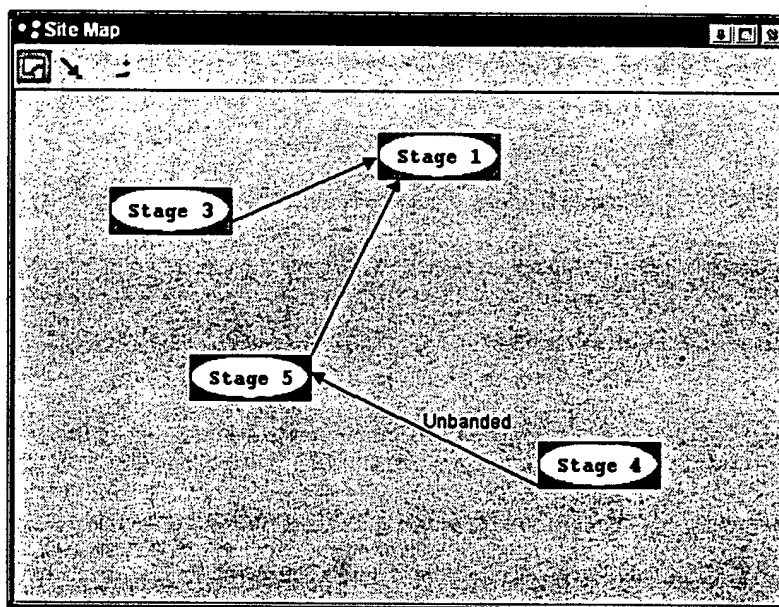


Regardless of the source of the new stage they will be automatically added to the Site Map.

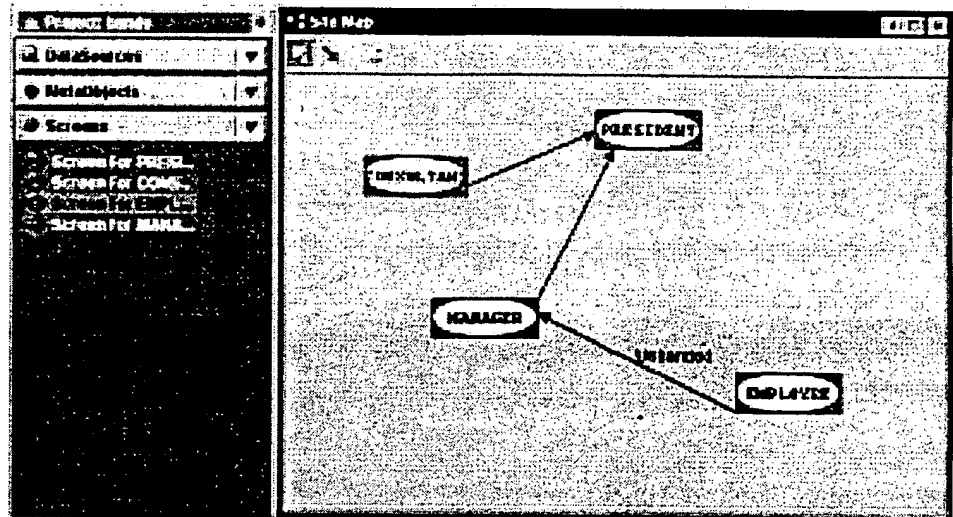


## Draw Arrow

The draw arrow allows you to form relationships between stages by connecting them.



You can also identify the stages in accordance with your particular environment by using the name field in the Properties dialog box.



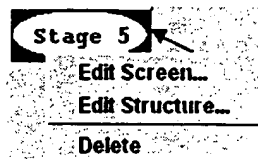
In this instance the names of the stages were changed to reflect an organizational chart.

A right mouse click on any of the stage boxes will open fields which allow you to edit the screen (using the Context Editor) or edit the structure (using the Structure Designer).

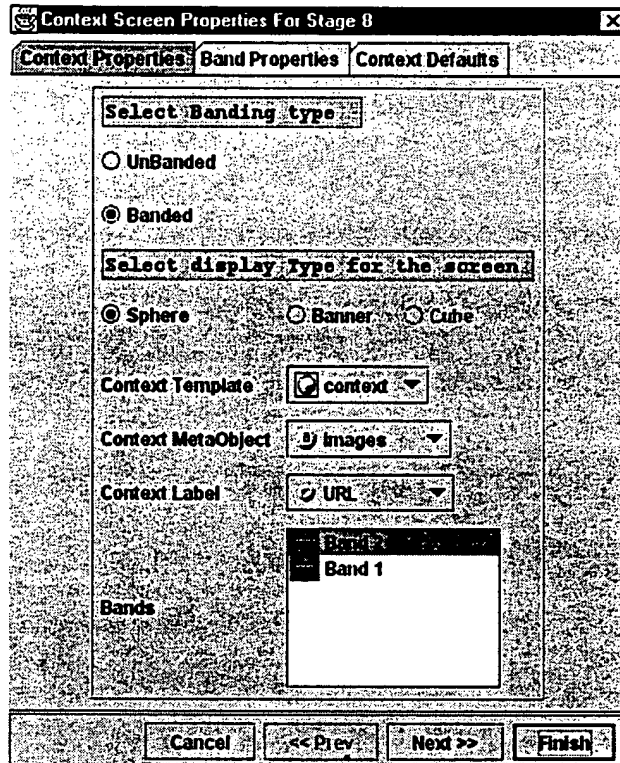
You can change the structure template you are using in that stage with the Properties dialog box.

These are basically drill-downs from one content to another when viewed by the AltoClient.

You can also delete the stage entirely.



In this example, Stage 8 was selected. Edit Screen was highlighted and clicked. This means that the Context Screen Properties dialog box for Stage 8 was accessed.



The processes associated with the Context Editor as well as Edit Structure has been previously presented and you are referred to those sections.

Congratulations. You have completed the development of your project.

Save your project and exit AltoStudio. AltoClient will be used to load the information you have just developed and present it to its user interface.

---

# *Index*

---

## **Symbols**

11

### **A**

AltoCube 13  
AltoServer 10, 12  
AltoSphere 13  
AltoStudio 10  
applet 20  
applets 15

### **B**

background colours 117  
browser 20  
browser plug-in 15

### **C**

Child 80  
Content 12  
Context 12  
cube 11

### **D**

default number of projects 52

### **E**

Enterprise (ERP) systems 18

### **F**

facets 12, 13  
foreground colours 117

### **H**

HTML/XML format 11  
hyperlinks 21, 72

### **J**

Java 15

### **M**

Master 80  
menu bar 23  
MetaClusters 13  
MetaData 12  
MetaObjects 21, 73

Multiple Document Interface 23

**P**

project 26

Properties 34

**Q**

Query section 21

Quick Access tool bar 23

**R**

RDBMS 18

**S**

sphere 14

Structure 12

**T**

T3 44

Test button 62

three-tiered technology 10

tree 12

tri-pane 15

**U**

URL 16

**V**

Version 31

**W**

WebLogic 44

wizards 19





## *AltoClient*

### **User's Guide**

AltoWeb, Inc.  
1731 Embarcadero Road  
Palo Alto, California 94303  
650.855.8880  
[www.altoweb.com](http://www.altoweb.com)

---

© 2000 AltoWeb, Inc.  
Printed in USA. all rights reserved.

This document contains proprietary and confidential information of AltoWeb, Inc. and is protected by Federal copyright law. The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form, in whole or in part, without the express written permission of AltoWeb, Inc.

The information contained in this document is subject to change without notice. Neither AltoWeb nor its employees shall be responsible for incidental or consequential damages resulting from the use of this material or liable for technical or editorial omission made herein.

---

# *Contents*

---

## **CHAPTER 1**

### ***Getting Started 5***

Introduction **6**

Welcome to AltoWeb **6**  
*Implementation 6*

AltoClient **7**  
*Three Layers Of Abstraction 8*

AltoStudio **12**

AltoServer **13**

AltoStudio Components **13**

Sections **15**

*Data Modeler 15*

*Layout Designer 15*

*DataView Designer 15*

System Requirements **16**

Definitions **16**

*AltoWeb Applet 16*

*MetaObject 17*

*MetaObject Connection 17*

CHAPTER 2

***AltoClient Basics* 19**

The AltoClient Graphic User Interface 20

Working with AltoClient 21

*Entering Information* 21

At Start-up 21

*Launching AltoClient* 21

AltoController Transport Controls 22

*Needle-and-ball* 24

AltoController Controls 26

*Swiping* 28

AltoClient Keyboard Controls 28

*AltoSphere Size* 28

*AltoSphere Rotation* 28

Keyboard Shortcuts 29

INDEX

33

---

This chapter presents an overview of AltoClient. Advanced information about the AltoStudio and AltoServer will be found in subsequent documentation.

This chapter includes the following topics:

- "Introduction" on page 6
- "Welcome to AltoWeb" on page 6
- "AltoStudio Components" on page 14
- "Sections" on page 15
- "System Requirements" on page 16
- "Definitions" on page 16

---

## *Introduction*

At the highest level, the objectives for AltoStudio are:

- To access, relate, and structure information from diverse databases, web files, and multimedia servers
- To apply visual properties to this data
- To provide a highly customized user interface which enables end users to navigate and interact with this data in a straightforward and efficient manner

---

## *Welcome to AltoWeb*

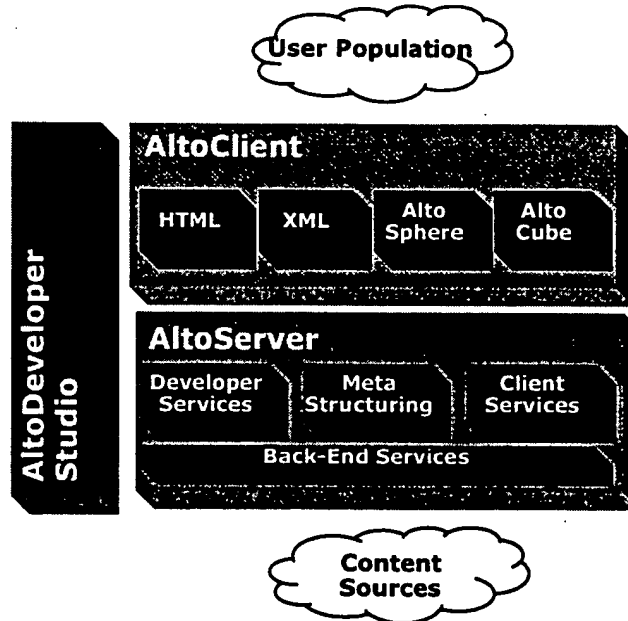
AltoWeb has developed a three-tiered technology which, at the user level, is reflected by an innovative user interface termed the AltoClient. The technology that underlies the AltoClient and makes it possible is two-fold: AltoServer and AltoStudio.

## **Implementation**

The implementation of the objectives of the AltoStudio must begin with the goal of determining just what, exactly, the composition of the user interface should be.

The issue is to determine what information, textual as well as graphic, should be included in that interface and how it should be presented. These objectives are created within the AltoStudio and ultimately reflected by the AltoClient.

The overall AltoWeb technology and its interrelationships with the AltoStudio, AltoServer and AltoClient can be graphically shown in the following diagram.



---

## AltoClient

The bottom line for AltoWeb is the AltoClient: AltoWeb's graphic interface. AltoClient, as well as the developer and server technology that supports it — is impressive: as a visualization and extraction tool, it lets you see both the conventional overall design of a web site yet, at the same time, allows you to navigate, exactly and quickly, to the specific areas you are interested in.

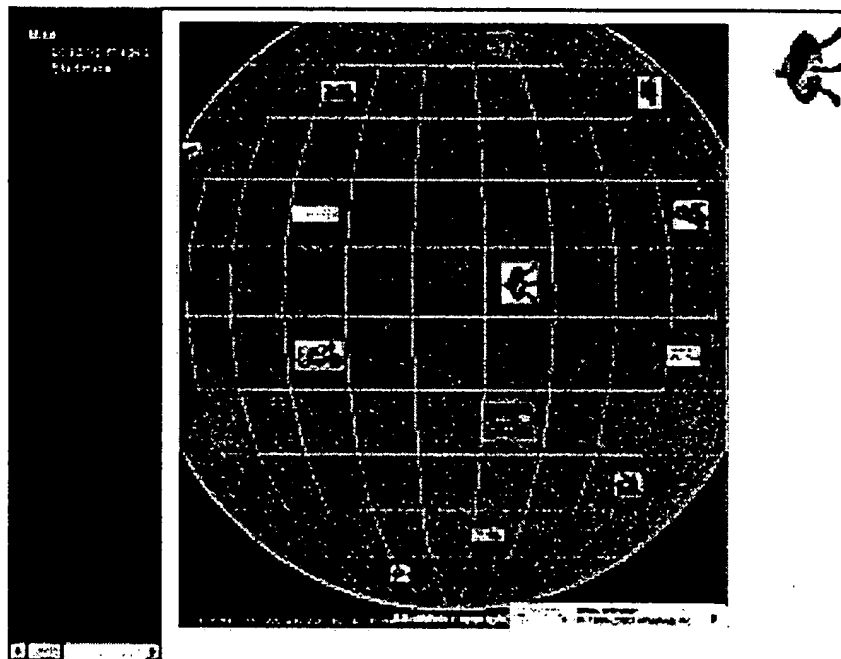
That is, instead of sifting through an interminable list of bulleted subsections and drop-down category bars, you can use a stylish, intuitive overview in the form of a three-dimensional sphere, cube, or HTML/XML format, to determine the specific information that you need.

AltoClient makes web sites easier to navigate by clearly laying out their entire structure. While simple organization charts, decision trees and box-and-line diagrams work for small sites, at larger sites they can often be tedious.

For example, you can spend a lot of time following link after link from a home page, through charts and diagrams, to ultimately find what you want. Or you can use AltoClient.

### Three Layers Of Abstraction

The AltoClient UI consists of three panes. Located from left to right they are: Structure, Context, and Content.





1. Structure — The hierarchical relationship of rendered facets to other facets is displayed in the left side frame.

Facet rendering is the visual display representing a structured link to additional hierarchical structures or terminal nodes of content.

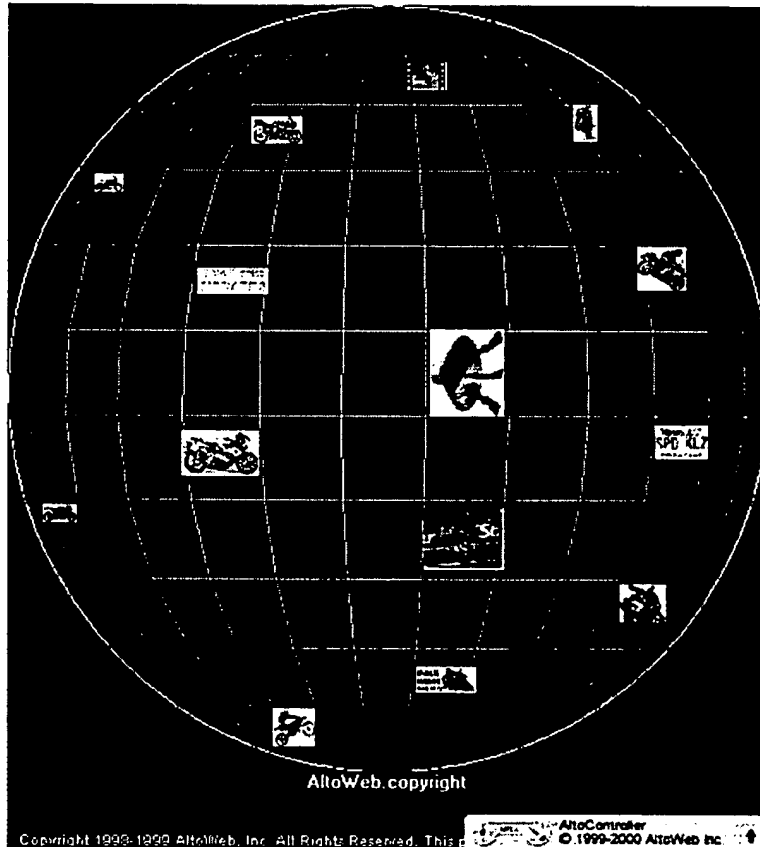


```
Mike
├── Loading images...
└── Start Here
```

The structure is similar in concept to a "table of contents" or master tree which represents the manner in which a web site has structured the content for browsing. In this case, structure is dynamically driven by the MetaData application previously determined in the AltoServer.

That is, the master tree structure pane is where MetaClusters are viewed.

2. Context — This is the center pane in the interface. It contains rendered capabilities including the oriented displays of facets and related linked contents.



There are several ways to render context including the AltoSphere (a spherical representation of facets), the AltoCube (a cubic representation of facets), or HTML/XML (a standard HTML/XML representation of facets).

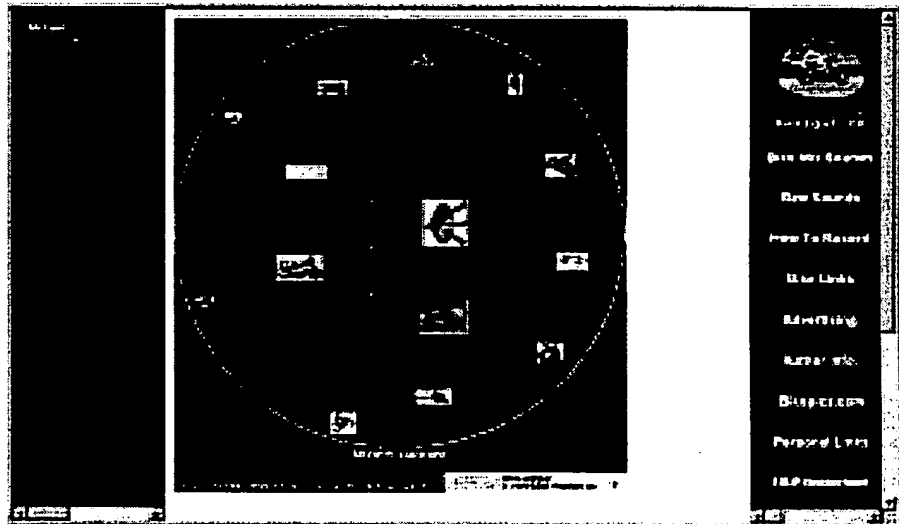
Placing your cursor on any visual representation of the rotating sphere will produce a pop-up dialog box which contains user-determined information about the contents of that image.

3. Content - The right pane, Content, represents all of the source information which has been refined by the structure and context panes.



In summary, AltoClient is the "front-end" user interface that incorporates a family of dimension-capable browser plug-ins or Java applets that maps query results onto 3-D shapes or HTML/XML documents. These documents, in turn, present web-site content in a unique tri-pane window that includes a user-determined spherical or cubic view for ease-of-navigation.

Although a specific interface will be user-determined to be specific to your environment, a typical AltoClient interface might resemble the following:



In this example, we have selected the URL *www.insweb.com* although any public domain URL could have been used. Just as importantly, many URL's could have been selected, with each occupying a separate facet on the sphere.

Specifically, the left pane (structure) shows you the contents of your selection. It represents a table of contents that are visually implemented in the center and right panes.

The center pane (context), a sphere in this case, consists of the information or URL sites (you can have many sites) you have transferred to individual facets on the rotating sphere.

The right pane (content), in this instance, shows the real-time contents to the URL site.

---

## *AltoStudio*

AltoStudio enables system administrators and Web site designers to:

- Create data models from multiple information sources
- To specify the operations and transformations that are to be applied to the data
- Customize the presentation of the data model for the end user

Using AltoStudio, you can map and customize the meta-data schema to an existing application and data source with minimal effort. This can be defined as software that allows the rapid development environment of content aggregation and configuration of the AltoClient.

AltoStudio can also be defined as a dynamic developmental environment from which you define data sources and, subse-

quently, forward those sources to web site developers for web page content.

In short, AltoStudio can be either a developer or a nondeveloper solution that is used to create the content and look of AltoClient.

---

## *AltoServer*

AltoServer is the cornerstone of the AltoWeb solution. It provides services such as connecting to data sources as well as loading and saving existing projects both for AltoStudio and AltoClient.

AltoServer provides the underlying technology that supports the connectivity to data sources. It both saves and loads projects for the AltoStudio and AltoClient.

AltoServer marshals data from distributed, heterogeneous data sources and forwards that information into a medium suitable for Web publication. It also leverages application server applications such as security database connectivity and naming services.

Examples of data sources that can be used by AltoServer include:

- RDBMS
- Video and other data streams
- Enterprise (ERP) systems

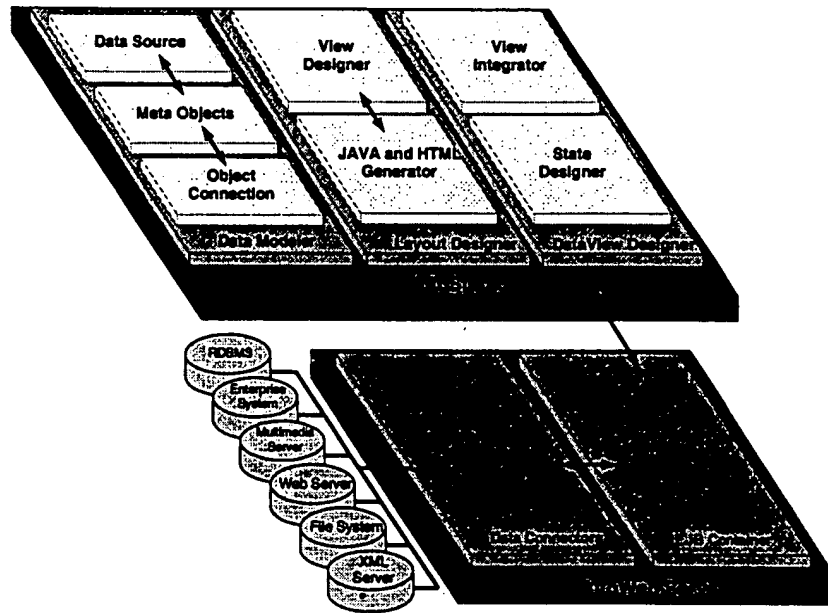
---

## AltoStudio Components

---

The overall relationship between the AltoStudio and the AltoServer can be shown in the following graphic.

**NOTE:** The site *MUST* be viewed with Shockwave in order to see all graphic and message elements.



---

## *Sections*

AltoStudio consists of three major sections:

### **Data Modeler**

The Data Modeler is used to:

1. Create connections to data sources by allowing you to specify:
  - Data sources
  - Paths to those data sources, and
  - Establish authentication parameters for the data sources
2. Define MetaObjects that contain references to data sources.
3. Create connections between MetaObjects which define relationships among them.

### **Layout Designer**

The Layout Designer incorporates template editors that allow you to create templates including:

- An overall view of the result set or data items
- The visual properties of each data item

### **DataView Designer**

The DataView Designer includes editors that allow you to:

1. Create connections between MetaObjects.
2. Establish the visual properties of the site.
3. Define the relationships among MetaObjects into a site map.

---

## *System Requirements*

System requirements will vary from platform to platform. In general, however, AltoStudio requires:

- Intel Pentium processor
- 2 MB hard drive disk space
- Windows 95, 98, NT 4.0
- JDK 1.1.7
- 64 Mb of RAM
- Microsoft Internet Explorer (4.0.1+)
- Netscape Navigator (4.0+)

---

## *Definitions*

Once AltoStudio has defined your solution, AltoClient allows you to navigate and interact with AltoWeb MetaObjects. The following definitions are specific to this process.

### **AltoWeb Applet**

An AltoWeb applet is a program written in the Java Programming language that can be included in an HTML page, much in the same way an image is included. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser.

Specifically, AltoClient runs as an applet inside a browser. And, as a client applet, it remains dependent on the AltoServer.



## MetaObject

A MetaObject is a descriptor to a specific connection containing selection criteria. It may also contain fields from a single data source including, for instance, information in a table or hyperlinks in an HTML document.

For example, a MetaObject might consist of a selection of all automobiles where the *location* = *germany*. Or a MetaObject may consist of all hyperlinks that start with *www.gullwing.com*.

MetaObjects can also have additional constraints including, for example, all *automobiles* = *gran touring* = *german* = *gullwing doors*.

## MetaObject Connection

The connections or relationships between MetaObjects is known as a MetaObject Connection.

A MetaObject Connection Query describes how two, or more, MetaObjects are related to each other.

The following Chapter: "AltoStudio Window" on page 21, begins the process of developing the AltoWeb solution.

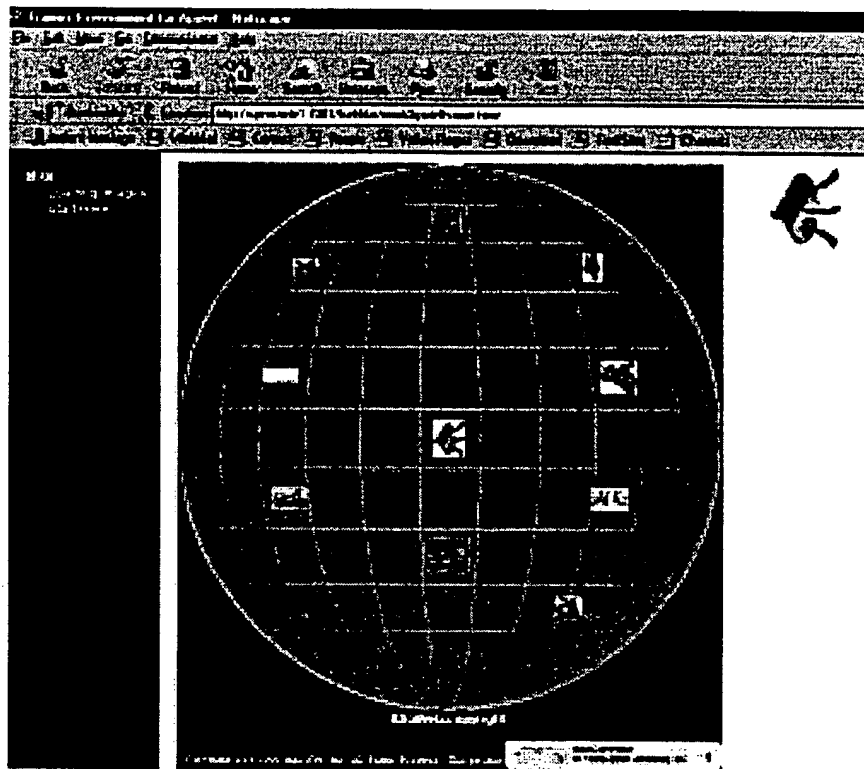
[illegible]

---

This chapter explains the form and function of the AltoClient user interface. This chapter includes the following topics:

- "The AltoClient Graphic User Interface" on page 20
- "Working with AltoClient" on page 21
- "At Start-up" on page 21
- "AltoController Transport Controls" on page 22
- "AltoController Controls" on page 26
- "AltoClient Keyboard Controls" on page 28
- "Keyboard Shortcuts" on page 29

Although the interface can be user-determined, a typical interface might resemble the following:



---

## *Working with AltoClient*

There are several conventional operational standards you should review before launching AltoClient.

### **Entering Information**

After you enter information into a text field on any AltoClient panel, you must either:

- Press the **Enter** key
- Press the **Tab** key

---

## *At Start-up*

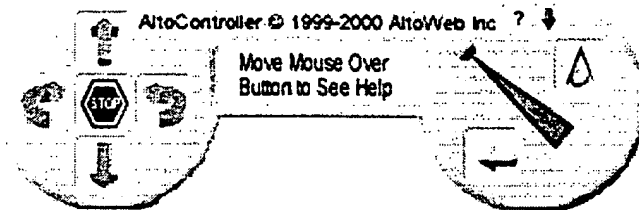
You must have your AltoStudio and AltoServer systems installed and configured before you can run AltoClient. Refer to the *AltoServer Installation Guide* and *AltoServer Developer's Guide* as well as the *AltoStudio Installation Guide* and *AltoStudio User's Guide* for assistance.

### **Launching AltoClient**

To launch AltoClient, simply access the directory where your system administrator has placed AltoClient. Normally, AltoClient can be accessed with a URL.

## *AltoController Transport Controls*


The AltoClient has its own set of cursor-activated transport controls contained in compact feature called the AltoController.



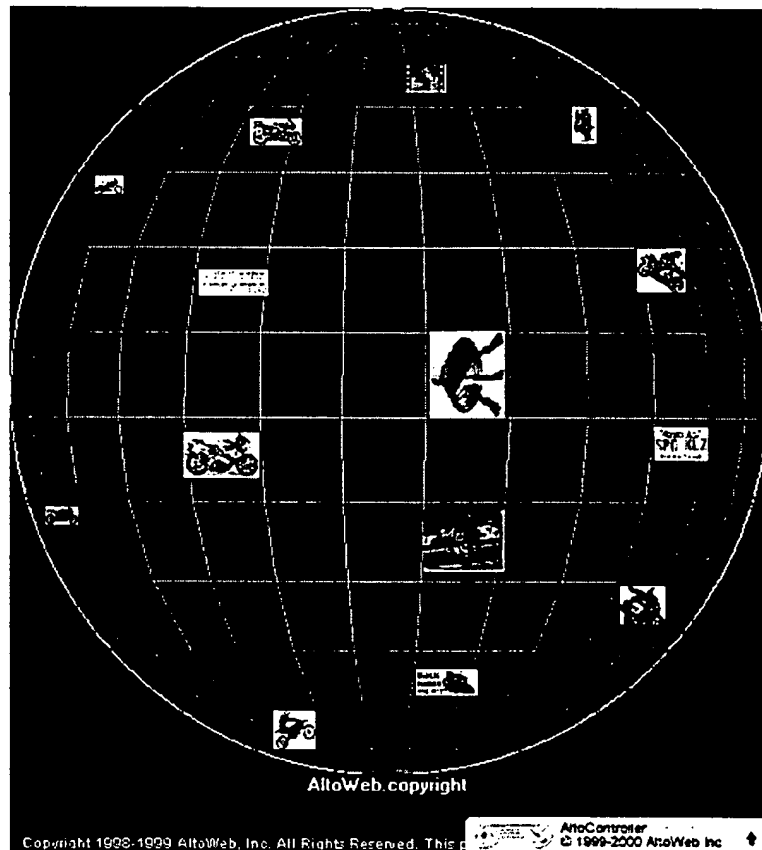
The AltoController allows you to determine the direction of rotation as well as to start, stop, rotate, and even resize the sphere itself.

Note that, while this discussion is particular to the AltoSphere, it applies to the AltoCube interface as well.

Additionally, if you are ever uncertain about the functionality of a particular arrow or function, the information box describes that function.

When it is not in use, the AltoController can be minimized by clicking the down arrow icon  located in the upper right-hand section of the AltoController.

A minimized AltoController context pane is shown in the following graphic.



Clicking the up arrow, restores the AltoController.

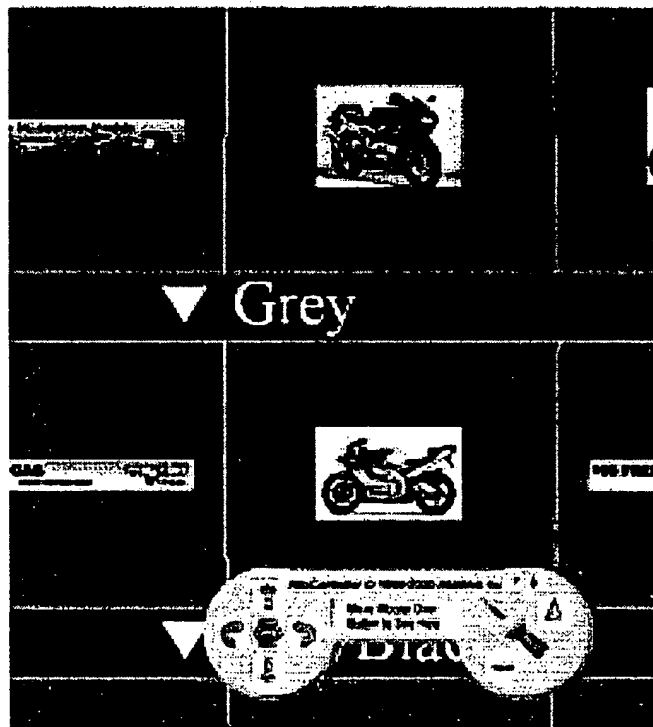
## Needle-and-ball

The primary tool for controlling the size of the sphere itself is the needle-and-ball.



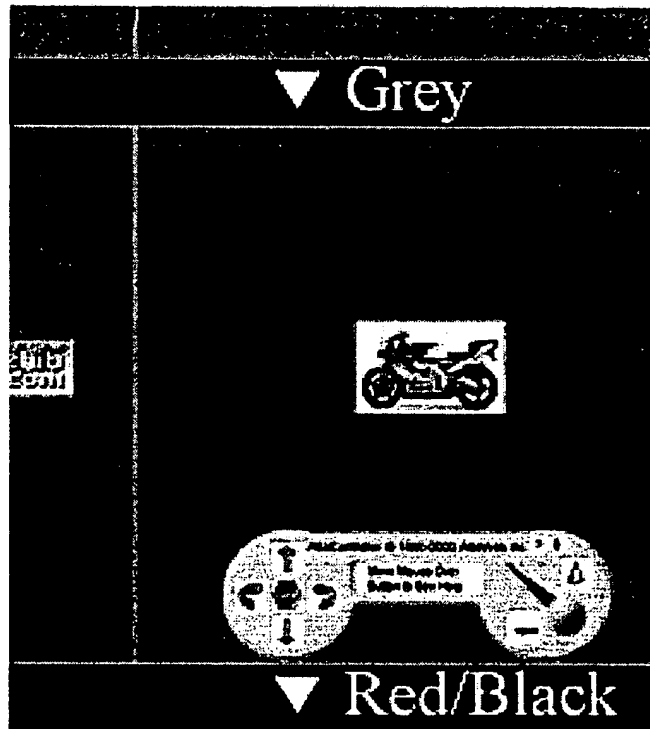
The needle-and-ball is a graduated linear slide. By placing your cursor on the ball and sliding it down the needle the sphere will increase in size.

For example, if you slide the ball to the midpoint on the needle you might have the following:





And, if you slide the ball to the limit of the needle you would have the following:



Note that the ball itself increases in size according the degree of magnification of the sphere.

## AltoController Controls

The following table list summarizes the other functions of the AltoController.

**TABLE 1. AltoController Functions**







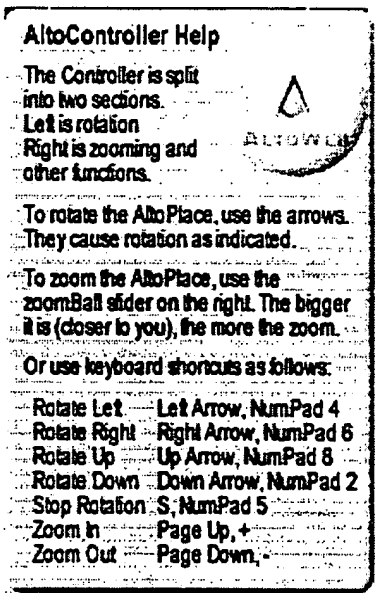


Function	Effect
	Rotates the sphere clockwise. Hold the arrow down to go faster.
	Rotates the sphere downwards. Hold the arrow down to go faster.
	Rotates the sphere counterclockwise. Hold the arrow down to go faster.
	Rotates the sphere upwards. Hold the arrow down to go faster.
	Provides direct access to AltoWeb's web page.
	Returns to the previously viewed stage.

TABLE 1. AltoController Functions (Continued)

Function	Effect
?	<p>Opens a help menu.</p> 
	Stops the rotation of the sphere.
	Minimizes the AltoController.

## Swiping

Swiping is a short cut. Just quickly move - while pressing and holding the *right* mouse button - the cursor in any direction you want on the sphere or cube. The sphere or cube will rotate in that vectored direction.

For example, if you move your cursor right (090 degrees or 3:00am/pm) the sphere will follow.

---

## *AltoClient Keyboard Controls*

### AltoSphere Size

Pressing the *Page Up* or *Page Down* keyboard buttons will either increase or decrease the size of the AltoSphere.

- Page Down = Zoom Out (will make the sphere smaller)
- Page Up = Zoom In (will make the sphere larger)

### AltoSphere Rotation

The speed of the rotation of the sphere is given, it cannot be either increased nor decreased. It can, however, be started or stopped.

- Pressing the **S** keyboard key will stop the rotation of the sphere
- Clicking the -> arrow key will rotate the sphere to the right
- Clicking the <- arrow key will rotate the sphere to the left
- Clicking the ^ arrow key will rotate the sphere up
- Clicking the v arrow key will rotate the sphere down

---

## Keyboard Shortcuts

Once you have loaded your image, you can control it through the interface itself using the AltoController and the mouse.

Or . . . you can use keyboard shortcuts.

The following table list summarizes the keyboard shortcuts.

**TABLE 2. Keyboard Shortcuts**

Shortcut	Effect
S	Stop the rotation of the sphere
Page Up	Zoom out - sphere will become smaller
Page Down	Zoom in - sphere will become larger
->	Rotate right
<-	Rotate left
^	Rotate up
v	Rotate down

[illegible]

---

## Symbols

7

---

### A

AltoCube 10, 22  
AltoServer 6, 9  
AltoSphere 10, 22  
AltoStudio 6  
applets 11

### B

browser plug-in 11

### C

Content 8  
Context 8  
cube 7

### E

Enterprise (ERP) systems 13

### F

facets 9, 10

### H

HTML/XML format 7

hyperlinks 17

### J

Java 11

### M

MetaClusters 9  
MetaData 9  
MetaObjects 17

### Q

Query section 17

### R

RDBMS 13

### S

sphere 10  
Structure 8

### T

three-tiered technology 6  
transport controls 22

U  
URL 12

[illegible]





***AltoServer***

**Developer's Guide**

AltoWeb, Inc.  
1731 Embarcadero Road  
Palo Alto, California 94303  
650.855.8880  
[www.altoweb.com](http://www.altoweb.com)

---

005090-2740000

© 2000 AltoWeb, Inc.  
Printed in USA. all rights reserved.

This document contains proprietary and confidential information of AltoWeb, In. and is protected by Federal copyright law. The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form, in whole or in part, without the express written permission of AltoWeb, Inc.

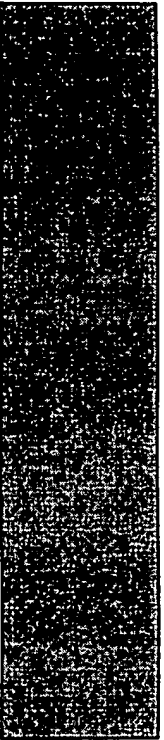
The information contained in this document is subject to change without notice. Neither AltoWeb nor its employees shall be responsible for incidental or consequential damages resulting from the use of this material or liable for technical or editorial omission made herein.

---

# *Contents*

---

<b>CHAPTER 1</b>	<b>Overview 9</b>
	Introduction 9
	<i>Unified Programming Model</i> 10
	Compatibility 10
	Application Environment 11
	<i>AltoClient</i> 11
	<i>AltoStudio</i> 11
	<i>AltoServer</i> 12
	AltoWeb Functional Block Diagram 13
	System Requirements 14
	<i>AltoStudio</i> 14
	<i>AltoClient</i> 14
	<i>AltoServer</i> 14
<b>CHAPTER 2</b>	<b><i>Programmer's Guide</i> 15</b>
	Developer Session 15
	Design and Development 16
	MetaObject 17



---

---

<i>MetaObject Properties</i>	18
<i>Define Logic</i>	19
<i>Define Queries</i>	19
<i>Create Connections</i>	19
<i>Template Editor</i>	20
<i>Context Editor</i>	21
<i>View Screens and Site Map</i>	21
<i>Save the Project.</i>	22
Additional AltoStudio Features	22
<i>Properties</i>	22
<i>Structure Editor</i>	22
<i>Insert Menu</i>	22
Summary	23

## CHAPTER 3

<i>Product Architecture</i>	25
AltoWeb Servlets	26
DataConnectors	26
EJB Container	27
<i>Entity Beans</i>	27
<i>Session Beans</i>	27
AltoWeb MetaCluster	28
Security	28
AltoStudio Visual Development	29
AltoClient Modeling Process	29
Application Development	30

## CHAPTER 4

<i>Theory of Operation</i>	31
EJB Session Interface	31
Client Connections	33
QuerySession	34
<i>The Query</i>	34
MetaClusters	35
MetaObjects	36
DataSets, Fields, and DataSources	37
<i>A Field</i>	37
<i>A DataSet</i>	37
QueryEvents	38
<i>DataInstances and DILists</i>	38

---

---

QueryActionLogic **39**  
EJB SuperClass **40**  
    *Projects* **41**  
Projects **42**  
ObjectConnections **43**  
    *Parent and Child Objects* **43**  
DataSet, Field and DataSource Objects **43**  
    *Field Objects* **44**  
    *DataSource Objects* **44**  
Compound MetaObjects and Joins **45**  
XML **46**  
    *XML Implementation* **47**  
MetaObject Creation and Mapping **49**  
Custom Logic Creation and Editing **49**

## CHAPTER 5

*Diagrammatic Illustrations* **51**  
The Structure of Classes in a Project **52**  
The Content of a Project – Data **53**  
Implementation of DataObject Interfaces for RDBMS **54**  
Data Mapping in the Application **55**  
Flow of Query Execution **56**  
Flow of Execution **58**  
Flow of Client Request to the Server **59**  
AltoServer Application **60**

## CHAPTER 6

*AltoServer Packages* **63**  
AltoServer Beans Super Package **63**  
AltoServer Application Super Package **65**  
AltoServer EJB Implementation **66**  
WebLogic Properties File **67**  
    *User Generated Entries* **67**  
    *Connection Pool for Testing Performance of Beans* **68**  
AltoWeb Server Properties **69**  
    *Projects Data Storage and Update* **69**  
    *XML Output* **70**  
    *Java Code Generation* **71**  
    *Compile command* **71**

---

*Code Generation for Logic 72*  
*HTML Generation 72*  
*Security 72*  
*Accessible Directories In Your File System 73*  
*Restricted Domains 73*  
*Data Sources 73*  
*A Limit on the Number of Results in a Query 74*  
*DataSource Specific Limits on Query Result Size 74*  
*Logic Classes 74*  
*Cache Management 74*

## **CHAPTER 7**

### **API Reference Classes 77**

**Parameter Descriptions 78**

**QuerySession 80**

*Description 80*

*Methods 81*

**QueryEvent 91**

*Description 91*

*Methods 92*

**DataInstance 93**

*Description 93*

*DataInstance 94*

*DI List 94*

*Methods 95*

**DIList 99**

*Description 99*

*Methods 99*

**DataSource 100**

*Methods 100*

**DataSet 114**

*Description 114*

*Methods 115*

**Field 120**

*Description 120*

*Methods 120*

**BasicQuery 124**


*Description 124*

*Methods 124*

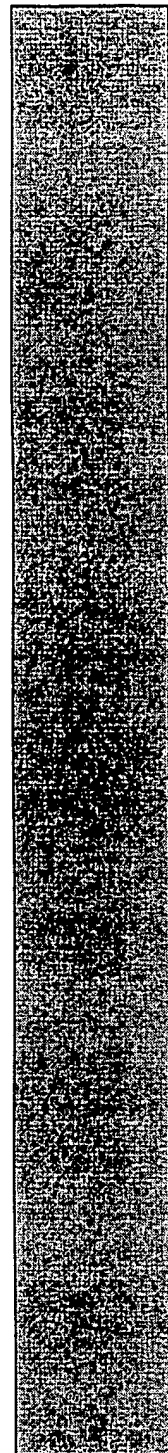
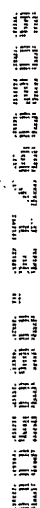
*Data Types 125*

**DynamicQuery 126**

*Description 126*



## Glossary 153





---

At the highest level, the objectives for AltoWeb is to provides an intelligent enterprise server and media navigation software solution that fundamentally enhances and broadens Internet application placements.

---

### *Introduction*

In order to implement this objective, AltoWeb provides the following capabilities:

- It transforms web sites into intelligent applications
- It provides real time structuring of all information constructs
- It provides superior navigation with dimensional (3-D) browsing
- It provides a next generation application server

AltoWeb creates web sites and applications that vastly extend the information utilization of the Internet, Intranet and Extranet environments. It creates entirely new applications in information processing environments.

AltoWeb also simplifies data access and manipulation, makes data transformation easy and, working with AltoStudio, minimizes development time and code size.

This AltoServer Developer's Guide assumes you are familiar with Java and database programming. It is intended for persons who are familiar with the following:

- 3-tier software application architectures
- Database connectivity
- Application server technologies

### **Unified Programming Model**

Nominally, this Guide documents the EJB Session Interface of the AltoServer.

AltoWeb provides a unified programming model that manages data from disparate sources. It hides the complexity of working with different types of data sources behind a graphical user interface (AltoStudio) and a simple programming API embodied in this document.

---

### *Compatibility*

AltoServer is intended to run on the current generation of application servers including BEA's WebLogic®, IBM's WebSphere® and Sun/AOL's i-Planet®, extending their web performance and application capability.

The initial product release runs on the web-based application server, BEA WebLogic 4.0.

---

## *Application Environment*

The AltoWeb Solution consists of three parts:

1. AltoClient
2. AltoStudio
3. AltoServer

### **AltoClient**

AltoClient can be either:

- An applet accessible through a web browser, or
- A browser plug-in an application

### **AltoStudio**

AltoStudio is a visual development environment which allows a web designer to define and customize web pages and applications. It works in conjunction with the AltoWeb Server.

AltoStudio creates the visual and functional elements which correspond to the business logic of the application. It connects the diverse data sources which make the project work.

You do not need complex software engineering skills and knowledge to use AltoStudio, but you do need to understand the concepts involved.

Alto Studio does the following:

- It permits introspection, connection and navigation of data sources.
- It creates objects that refer to data fields within the data sources.
- It creates relationships and logic between the objects in the applications.
- It creates templates to view information.
- It maps objects to visual tags within the templates.

In short, AltoStudio defines the application which then runs on the AltoServer.

## **AltoServer**

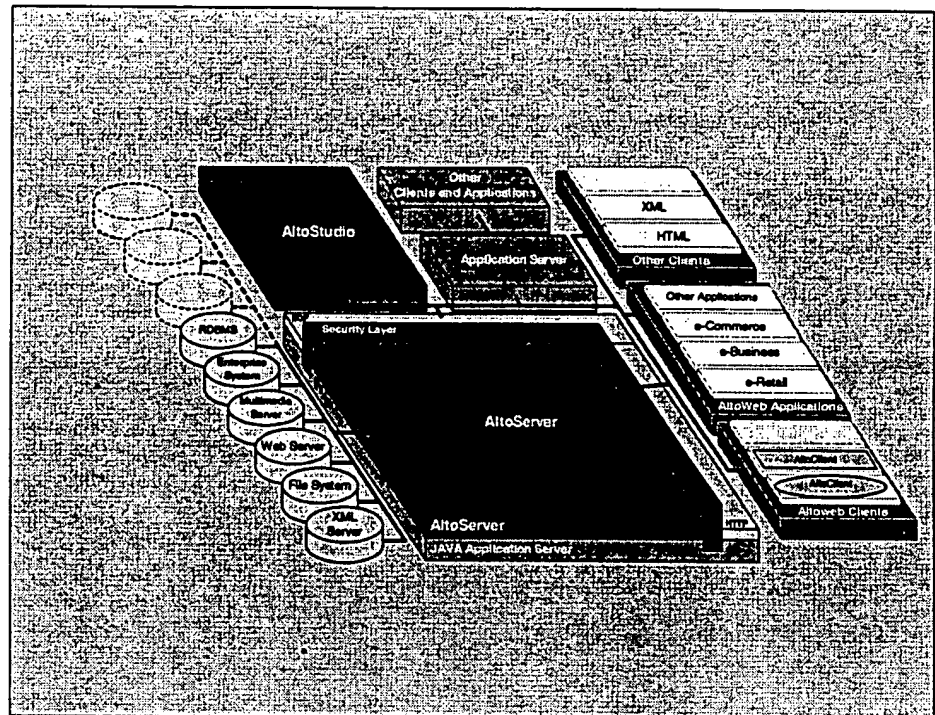
AltoServer provides the framework that defines and structures the relationships between application web pages and data sources.

While it runs on an application server, it permits the creation and editing of hierarchical data structures and search strategies for different applications.

---

## *AltoWeb Functional Block Diagram*

The following diagram shows the functional relationships between AltoClient, AltoStudio and AltoServer. It shows their relationship to clients and data sources.



---

## *System Requirements*

System requirements will vary from platform to platform. In general, however, the following details the requirements:

### **AltoStudio**

- An Intel Pentium II processor or greater
- 2 MB hard drive disk space
- Windows 95, 98, NT 4.0, UNIX, Linux
- JDK 1.1.7B, or JDK 1.2.3
- 64 Mb of RAM
- Microsoft Internet Explorer (4.0.1+)
- Netscape Navigator (4.0+)
- Symantec Visual Café 3.0 or greater

### **AltoClient**

#### Pentium II

- 32 MB RAM or higher
- Microsoft 95/98/NT 4.0 or higher
- Navigator 4.0 or higher/Explorer 4.0 or higher

### **AltoServer**

- Pentium II
- 128 MB RAM or higher
- 15 MB Hard Disk memory or higher
- Microsoft Windows NT 4.0 or higher, UNIX, Linux
- JDK 1.1.7, or JDK 1.2.3
- BEA WebLogic 4.0 or higher, EJB compliant application server

---

The purpose of this chapter is to explain the AltoStudio application development process.

Please refer to the *AltoStudio User's Guide* for information about the AltoStudio visual development environment.

The user should install AltoStudio and AltoServer as described in the Installation Guide. Ensure that you pay special attention to the system requirements. Connections to clients and data sources should be implemented, as appropriate.

---

### *Developer Session*

Once AltoStudio and AltoServer are installed, connect AltoStudio to the AltoServer. This establishes a Developer Session.

Enter the server name, login and password.

Click **Connect**.

AltoStudio verifies that a connection is established.

To use the Programming API described in this document, you must create or open a Project using AltoStudio, even if it just consists of default data.

Save the Project with a new Project name.

---

## *Design and Development*

The AltoStudio design and development process consists of the following steps:

1. Create a New Project.
2. With the connection to AltoServer established, open the File menu and select either Open a Project or New Project. Existing projects are available from the directory structure. Projects consist of one or more MetaObjects.
3. Connect to the Data Source(s).
4. With a Project opened, click on the DataSource panel.
5. The DataSources are divided by Type. These are FileSystem, SQL/JDBC database, Web Server and XML.
6. Select the appropriate type. The system lists the data sources that may already be a part of this project.

When selecting a new data source, the system prompts the user to set the Connection Properties. This essentially involves specifying the source URL or file location (in the case of a File System). You can also specify the login and password for access to the source.

Bear in mind that you connect to these four types of sources which may have quite different formats and disparate rules for obtaining and processing data. The system provides the means to test the source, and in the case of a fault or inappropriate system, you can default or re-select the source.

7. Click **Finish** to select the DataSource.
8. Create the MetaObject or MetaObjects.



---

## *MetaObject*

The MetaObject is an atom of business logic. It represents the request for information and embodies the purpose of the data access. In short, it is the reason for doing the application.

Typically, you access a table, a sequence or a specific data source and set parameters to access the data from the client.

The MetaObject also defines the data source(s) where you obtain data. You specify related tables to serve the user and visual tags to connect to visual elements in the client screen presentation. You may specify a sequence of different screens with links between them. The MetaObject also allows you to program logic and queries.

Each MetaObject embodies a definition of the client-server information access.

You can select three types of MetaObjects.

1. Compound MetaObjects

Compound MetaObjects are formed from two or more pre-existing MetaObjects. They are constructed from a Master MetaObject which represents the key control data in the resulting MetaObject and other MetaObjects, embodying Fields as selected from the MetaObjects.

2. Custom MetaObjects

Custom MetaObjects are formed from first principles. These principles are from data structures found in the data sources.

3. Linked MetaObjects

Linked MetaObjects are formed from pre-existing MetaObjects which refer to different DataSources. There is said to be a link which defines the data access parameters for each new MetaObject.

When you click on the MetaObject name or symbol, you create a new MetaObject.

## MetaObject Properties

You define the MetaObject properties when you create a new MetaObject. Some property windows are excluded by the type of MetaObject selected.

MetaObjects are constructed from data structures in DataSources. These may be represented as DataSets which map between data in more than one data structure. DataSets are tables consisting of fields (columns).

To create a MetaObject is to create an object that consists of fields.

- **Name and Type**

Allows you to create a MetaObject from a DataSource, from other MetaObjects, by link to another MetaObject, or "Just Create One." Enter a name to create a new MetaObject.

- **Choose Sources**

Allows you to select or deselect the DataSources associated with the MetaObject

- **Define Joins**

Allows you to define the Master and Child DataSets that will form a Compound MetaObject. You also define Join Type i.e., Inner, Outer, Multiple Joins.

- **Define Fields**

Allows you to define the Field. The field consists of the Field Name, Origin, Origin Field, Type and Visual Tag.

The Field Name is a name that you give to the Field for inclusion in this MetaObject.

The Origin is the name of the DataSet where it resides in the DataSource selected.

The Origin Field is the field name in the DataSet.

Type defines the data type to be used in the MetaObject.

The Visual Tag is the definition of how the Field is represented by the Context Editor.

By defining each Field and giving it a name, and by clicking **Finish**, you add MetaObjects one by one to the MetaObject which shows up under MetaObject in the Project.

### Define Logic

This feature allows you to define Java code logic that is performed on the result of a query. You can define any logic (i.e., mathematical and symbolic logic operations) that is allowed by the Java language either sequentially or in loops (iteratively). You can also name code that performs customized applications unique to your operations.

### Define Queries

This allows you to perform specialized queries on data sources. At the time of this release, this feature is limited to simple SQL defined queries. That is, the Query Type is currently only Simple SQL. The Query parameters, the program, are listed.

Give the Query a name so that you can reuse it.

When defining MetaObject properties, you can, at any time, view the actual data in the data source by clicking Show Data. This also will show you the results of operations that you have defined such as the joining, logic or querying operations.

### Create Connections

The Relationship Diagram is used to create connections or to specify links between data in your application.

When you have created one or more MetaObjects, you can create relationships between them so that when you access your application (using one of the visual tags that represent it), you will go to the MetaObject connection that has been defined in the Relationship Diagram.

The process is straightforward:

1. Select the MetaObject or MetaObjects that you want to connect.
2. Drag the MetaObject or MetaObjects onto the Relationship Diagram.
3. Click on the line icon.
4. Draw a line or lines connecting the MetaObject.
5. Click on the line connection(s) to define the nature of the link.

For example, if you want a specific field of a MetaObject to be displayed when another field of a different MetaObject is clicked, define the connection in the dialog box. Connection may be a simply one-to-one MetaObject, or as many as defined in the dialog box.

Alternatively, a circular single connection can define a MetaObject that access its own Fields.

## Template Editor

The Template Editor is used to create Templates and Visual Tags.

It allows you to define the information that appears in the facets and pop-ups using Visual Tags. You name the Visual Tags in the Template Editor and enter their name in the Define Fields dialog box when creating a MetaObject.

There are four types of visual tags:

- Text box
- Number box
- Date box
- Image box

By selecting the appropriate box, you can also move the boxes forward, backwards, to front, to back, centers vertically and horizontally, and align top, bottom, to left or to right.

You can enter text into each box as well as size and position each box within the window. You can enter single line text, multiple line text, numbers, dates and images into the visual tag boxes.

## **Context Editor**

The Context Editor is used to create screens and to design the visual composition and functionality of the client context screen.

It allows you to create the Context screen. The Context Editor shows a circular profile representing the sphere in the AltoClient window. It represents bands and facets which can be defined in terms of the visual tags which relate to Fields, as defined in the Define Fields window, when you were creating a MetaObject.

You can add or delete bands and pop-ups and assign them to MetaObjects and Fields. The bands may cover one another and are layered. A single click on each band reveals the next layer together with its functionality. As you click on each band, a popup appears into which you can enter text, images or URL data. There is one popup per band. You can move the position of the horizontal lines which define the bands by clicking the arrow keys.

## **View Screens and Site Map**

When you create Context and Template information, you also create Screens. Screens, in turn, are defined with reference to Stages i.e. Stage 1, Stage 2, Stage 3, Stage 4... where each Stage represents a sequence of a screen relationship and embodies the visual and functional interactions as defined therein.

By selecting screens, the various Stages created during the design process are listed in order. By opening the Site Map, you can see the current Stages and their relationships. If you want to move from Stage 1 to Stage 3, draw a line linking the two. This means that the client screens will change from one Stage to another. In other words, all the visual and functional relationships will change to a different class.

## **Save the Project.**

When you click **Save the Project**, the project is saved via the Developer Session to the AltoServer, where it can be accessed by the client.

---

## *Additional AltoStudio Features*

The following features are worthy of comment.

### **Properties**

The Properties window allows you to view and modify the properties of the selected item. Properties refer to bands, facets and pop-ups.

### **Structure Editor**

The Structure Editor, the left hand window of the client, is not implemented at this time. There is currently only one Structure window type available on the client, and the arrangement of Structure, Context, and Content as well as the dimensions and form of the windows are fixed at this time.

### **Insert Menu**

The Insert menu provides a convenient way to insert New Data-Source, MetaObject, Screen and Template files into a Project.

---

## *Summary*

It is possible to create a project by performing the steps listed above in a different order.

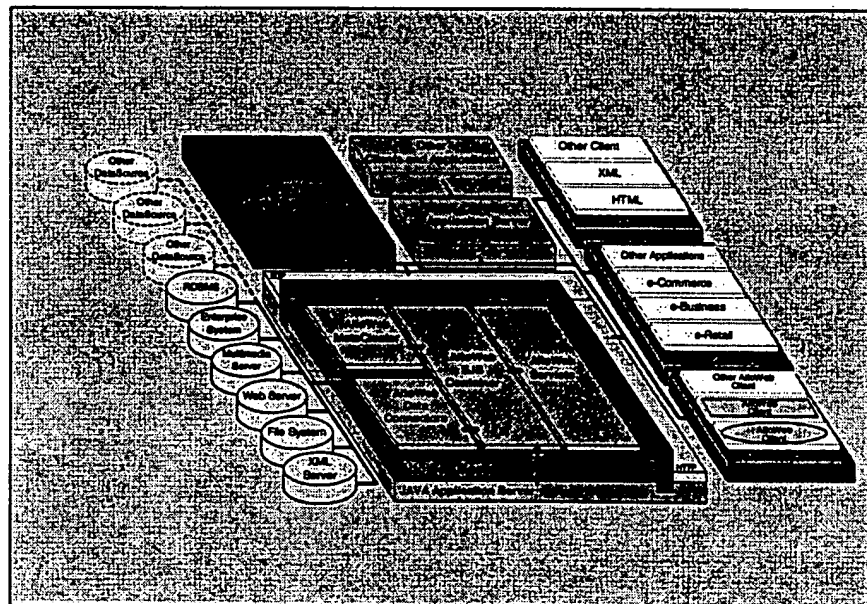
Some steps will not be necessary for certain applications. There is considerable flexibility in how to proceed, and the application can be designed by different individuals such as those skilled in networks, data content, visual design and so on, as appropriate.

It is easy to go back and make changes. It is also possible to save and compile the code on the spot, test and try out the application to see if it meets the design expectations.

Year	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	



To understand the operation of the system, you need to understand the Product Architecture. A block diagram of the system is shown below.



Basically, the AltoServer runs on Java Application Server such as WebLogic and is constructed from Enterprise Java Beans (EJBs), not to be confused with Sun Microsystems Java Beans. All programming is in the Java language.

AltoServer consists of the following components:

- AltoWeb Servlets
- AltoWeb DataConnectors
- AltoWeb EJB Container
- AltoWeb MetaCluster

---

### *AltoWeb Servlets*

AltoServer Servlets provide an interface to client and data base applications using the HTTP protocol. For each type of application, there is a different servlet. For example, there are servlets for HTML, XML, AltoClient (AltoServlet), eCommerce, eRetail, eBusiness, and other applications as illustrated in the Figure. Servlets provide interfaces to the application environment.

---

### *DataConnectors*

AltoServer DataConnectors provide the interfaces, the connections, to the sources of data. For each type of data source, there is a different data connector. For example, there are data connectors to RDMS, ERP, Multimedia Data Source, Web Data Source, File Data Source, and XML Data Source, as appropriate. The protocol is consonant with the data technology. Data connectors provide a transparent connection to the data sources.

---

## *EJB Container*

the AltoServer EJB Container controls enterprise beans and provides system level services for EJB's that run within the EJB Container. Session and entity beans provide functionality to the application.

The AltoServer EJB Container uses both entity and session beans.

### **Entity Beans**

An entity bean is used to represent underlying objects. The most common application for entity beans is the representation of data in a relational database.

A simple entity bean can be defined to represent a row in a database table, where each instance of a row represents a specific row. More complex beans could represent views of joined tables in a database where one instance represents a specific customer and all of that customer's orders or order items.

### **Session Beans**

While Entity beans are used to represent information that may be common to multiple clients, Session beans are used to allow state information both for the client(s) or for points in the database.

That is, a session bean is an enterprise bean where each instance is created through its home interface and is private to its client connection. A session bean instance cannot usually be shared with other clients. This permits the session bean to maintain the client's state.

Entity and Session Beans that are specific to AltoWeb include:

- DataView Entity Bean

DataView beans define the user's visual (or multimedia) experience in the application.

- DataView Session Bean
- Query Entity Bean

Query beans define hierarchical data structures and search strategies for an application. Query beans carry the data content communication between servlets and data connectors.

- Query Session Bean
- Developer Session Bean

Developer bean interfaces to the AltoStudio or to other Application Servers using the IIOP protocol. The Developer bean defines MetaObjects, which are descriptions of the applications

---

## *AltoWeb MetaCluster*

An AltoServer MetaCluster contains MetaObjects which consist of listings of data. This is the core of the application as they provide road map to the actual information, relationships among information items, and visual representations of the information

AltoServer EJBs support multiple client sessions and interfaces to MetaClusters.

---

## *Security*

AltoServer provides two forms of Security associated with the WebLogic application server.

- Authentication
- HTTP and secure sockets

---

## *AltoStudio Visual Development*

An application designer uses AltoStudio, which has the following four basic functions:

- It defines MetaObjects
- It defines relationships between MetaObjects
- It defines the MetaObjects to data source map
- It manages system performance

The AltoStudio visual development environment defines MetaObjects which are subsequently sent to AltoServer through the DeveloperSession bean. MetaObjects created by AltoStudio are formed into Projects which define AltoClient applications, usually thin client (<80 kB) applications.

Presently, prototype applications are typically web based involving access to databases such as corporate eCommerce, eRetail and eBusiness. They can provide access to desired information in only two or three mouse clicks compared to many clicks using current technology.

---

## *AltoClient Modeling Process*

The modeling process defines an AltoClient with the following characteristics.

- Visual information items such as Layout and DataView (structure, context, content screens)
- Data source information
- Relationships between data sources
- Relationships between data structures and how to connect to them

A more complete discussion of these subjects is included in the *AltoStudio User's Guide*.

---

## *Application Development*

Three packages are provided which permit application development at the AltoServer level. These packages allow the application designer various capabilities which are listed in detail in the Application Programming Interface (API). (It is possible, however, to go to the Java code to define other applications.)

1. The QuerySession package allows the user to run queries on MetaObjects.
2. The Meta package allows the user to define the application.
3. The Data package is a programming API which enables the connection of any DataSource to AltoServer.

The AltoServer API is a listing of all the necessary information at this level to perform application development.

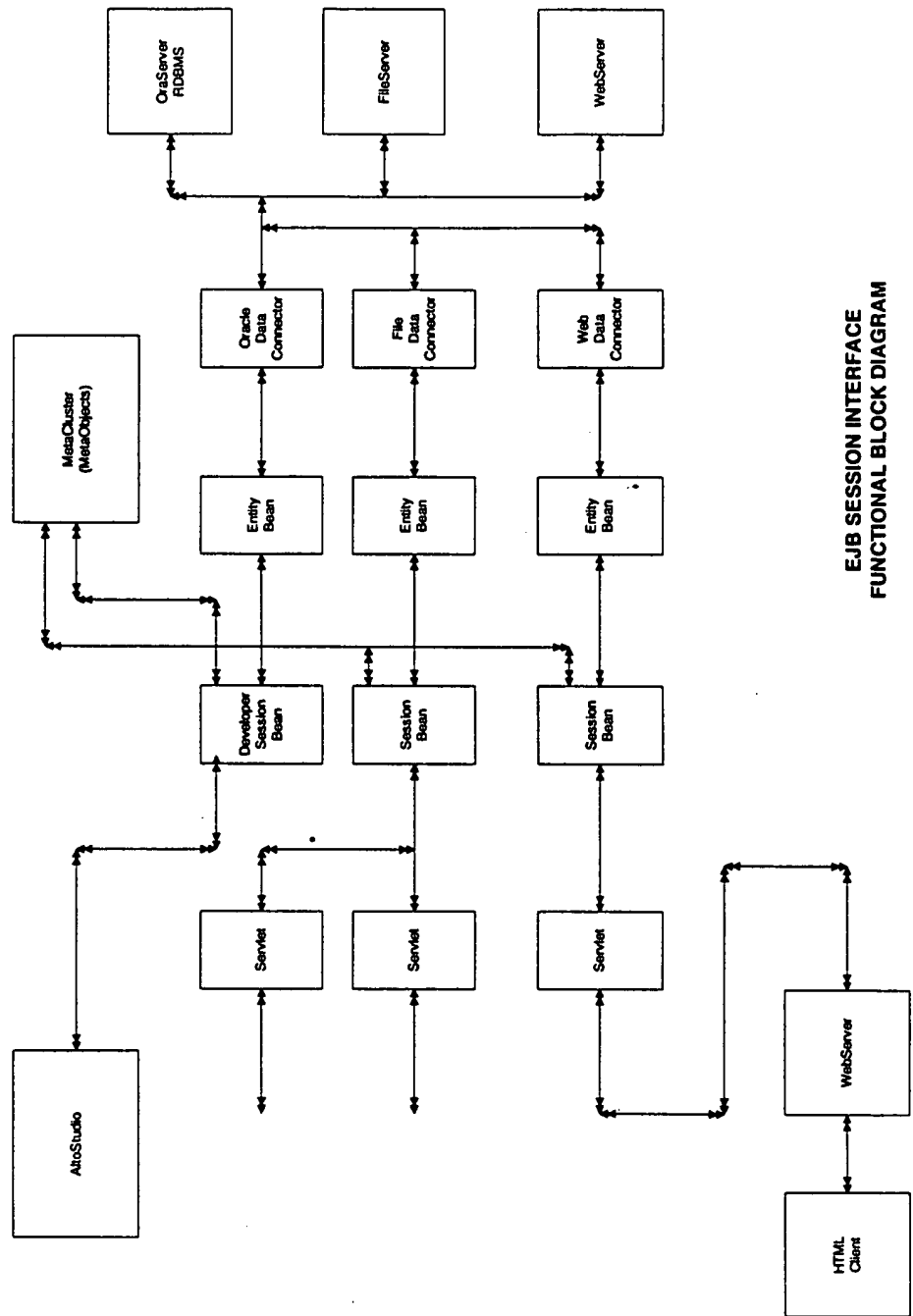
---

Theory of operation describes the EJB Session Interface: the requests and responses that occur as a result of the interaction between client and server.

---

***EJB Session Interface***

The Session EJB Interface is described by the EJB Session Interface Functional Block Diagram shown on the following page.



EJB SESSION INTERFACE  
FUNCTIONAL BLOCK DIAGRAM



The sequence of events or invocation flow is:

1. There is a request from an HTML client (or other client) to the WebServer via HTTP or another protocol.
2. There is a request from the WebServer to the appropriate servlet.
3. The servlet invokes a session EJB.
4. There is a query on a MetaObject. This invokes a QuerySession.
5. QuerySession finds the Project and MetaObject (from the MetaCluster).
6. Project and MetaObject information is embodied in an entity bean which represents the application.
7. QuerySession gets the QueryEvent of this MetaObject or uses the default QueryEvent.
8. QuerySession runs the query. It runs a DynamicQuery or ConfigurableQuery.
9. QuerySession accesses the DataSource file.
10. The DataSource sends results as a DataInstance or DIList.
11. The results (as DataInstance or DIList) are processed by QueryActionLogic (the Logic Model) which performs logic operations on the results.
12. The processed results are returned to QuerySession as a DataInstance or DIList
13. The results (DataInstance or DIList) are sent to the WebServer and to the client via the servlets.

---

### *Client Connections*

AltoServer uses servlets to connect to the client and to send requests to the QuerySession. these servlets are customized for different applications such as HTML, XML and AltoClient.

HTML servlets use the HTTP protocol.

Alternatively, there can be a direct connection from the client to the EJB interface using the IIOP protocol. In this case, items 2 and 3 above, in the Sequence of Events would not be applicable.

---

## *QuerySession*

A QuerySession is a stateless session program which resides on session beans. It does not keep information and it does not maintain state on behalf of the client.

QuerySession requests and receives data based on criteria for a query. This includes primary key reference, matching of items, range conditions, specified query, MetaObject name, property values, ObjectConnection, and parent/children of MetaObjects and/or DataInstances.

QuerySession also handles the processing and execution of queries including sending Dynamic and Configurable Queries and the receipt of DataInstances and DILists.

It also controls the flow of data through data connectors (i.e., program connections to different data sources).

## **The Query**

A query is a request for information which means that every query is independent of every other query.

A query is controlled by three Java classes:

- **BasicQuery**

The Basic Query interface handles standard Java data handling between data sources and the AltoServer.

Basic Query is a copy from the JDBC API. It converts data types to be compatible with AltoServer.

It should not need to be modified but is included in the API for reference.

- **DynamicQuery**

DynamicQuery is a request for information which can access a range of possible data including parameters, names, and parents, among others.

It is a stateless program. No information is kept.

It can return DataInstances and DILists by specifying Single and Many. Dynamic Query restricts the query to meaningful requests.

- **ConfigurableQuery**

ConfigurableQuery is an interface for data processing requests created by AltoStudio and contains predetermined requests for information.

AltoStudio defines ConfigurableQuery using the connection diagram which supports simple connections, parent/child connections, multiple connections involving fields and datasets.

It supports many data sources including SQL based DBMS. Queries can be written in SQL. The ConfigurableQuery can join, order and apply conditions and ranges.

A detail listing of Query features is given in the API.

---

## **MetaClusters**

The MetaCluster is a part of the AltoServer program which stores requests for information in the form of MetaObjects.

MetaObjects are created by AltoStudio as part of a Project. When saved by AltoStudio, it resides in the MetaCluster until called by a program — a query.

The Project, an association of MetaObject and ObjectConnections is also saved in the MetaCluster. A MetaObject hash table is a compendium of MetaObjects and ObjectConnections within a project that shows the composition of a project.

---

## *MetaObjects*

---

A MetaObject is an abstract representation of data to be requested. This may include lists of properties such as strings, numeric values, logic and connections to other objects. In short, it is a representation expressed as an object that exists independently of data sources. MetaObjects are used to map data demanded by the application.

In AltoServer, data sources are represented by DataSets.

In an RDBMS, a DataSet is a table or a mapping of data from one or more datasources, with columns and rows.

An introspection is a list of items that can be accessed from a data source.

The MetaObject selects the data it requires from one or more DataSets to form the programming instructions for a query. Similarly, other types of data source are mapped to different DataSet types. The data sets requested from a DataSet by a MetaObject is called a DataObject. It is predefined by a MetaObject. A DataObject defines the query from one or more data sources.

A MetaObject consists of the following items.

- an identifier – a unique name in the project
- properties – a list of Field objects from a DataSet
- content – an object that refers to the Field instance
- object connections – a member of the ObjectConnection class

A single instance of the MetaObject class always refers to Fields from a single instance of a DataSet.

An object that contains fields from two DataSet objects (two tables, tables in different schemas, different databases, etc.) can be created by connecting two MetaObjects.

AltoStudio creates a connection to DataSources or maps a new MetaObject to existing MetaObjects.

An object connection is a relationship defined between MetaObjects to define a data base connection.

---

## *DataSets, Fields, and DataSources*

A DataSet is a table or sequence which embodies a relationship between two or more variables and maps to one or more data sources. A DataSet consists of Fields.

### **A Field**

A Field is a single element of a DataSet. This may include a column or a row in a table. Although commonly found in RDBMS's, DataSets may be used to represent any underlying data source whether web, file, or multimedia. DataSets may be used, in principle, to map to any actual data source. One DataSet can be used to create several MetaObjects.

### **A DataSet**

A DataSource is an interface that encapsulates a source of data: the location of a source of data and the connection to it.

The DataSource class is configured by AltoStudio included configuration and MetaObject definitions from the underlying data source.

Its runtime functions create an actual connection to the data source. The DataSource permits connection to DataSets and Fields as well connections to nativeIdentifiers of the data source connection.

The user can select from items accessible to AltoServer. A data-sources hash table is a compendium of different data source locations either within individual or multiple sources. It is a reference source to permit routing of requests to the appropriate data.

AltoStudio is the key user source of introspection concerning data sources.

---

## *QueryEvents*

When a MetaObject is called by the QuerySession, it resides on an entity bean. The entity bean is stateful. It does not lose the information embodied in the MetaObject and continues to exist beyond the lifetime of the application.

A QueryEvent is a request to initiate a query. It is said to occur when the QuerySession sends out a query defined by a MetaObject or ObjectConnection from the MetaCluster. The QueryEvent abstract class can also be used to modify the query criteria.

During runtime, the system (AltoServer) executes requests. When a QueryEvent has occurred, the underlying data source is queried, and it sends a result.

## **DataInstances and DILists**

A result can have one of two different forms.

1. A DataInstance

A DataInstance is the result of a query consisting of objects: the capture of only one result, although the result may consist of many items requested by the MetaObject.

2. A DI List

A DIList is the capture of more than one DataInstance.

The following diagram shows the difference between a DataInstance and a DIList.

Project Name					
MetaObject Name					
Item 11	Item 21	Item 31	Item 41	Item 51	Item 61
Item 21	Item 22	Item 32	Item 42	Item 52	Item 62
Item 31	Item 23	Item 33	Item 43	Item 53	Item 63
Item 41	Item 24	Item 34	Item 44	Item 54	Item 64
Item 51	Item 25	Item 35	Item 45	Item 55	Item 65

A DataInstance is represented by the left hand column and consists of a Project Name and a MetaObject name plus results of the query as defined by the MetaObject. In this case, these are items 11, 21, 31, 41, 51, which simply bear a relationship to the MetaObject, but may have come from diverse and unrelated tables and sources.

A DI List is represented by all events in the table including the Project Name and MetaObject Name. It also includes all the items which embody six DataInstances. The columns, as represented here, can be addressed separately, as either numbered columns or as "Temp" items, so that results from a DIList can be re-formed to DataInstance data. ("Temp" items are an arbitrarily defined parameter in a DIList method.)

---

### **QueryActionLogic**

---

When the results are returned, they may be further processed by a logic model. The QueryActionLogic, associated with the Project and MetaObject on the entity bean, is executed as a program on the session bean. It takes the results (data) returned from the data source as DataInstances or DILists and performs logical

operations such as calculations and Booleans to provide an end-result that can be subsequently sent to the client.

The QueryActionLogic program can also re-organize the data to meet particular client presentation requirements.

Query Action Logic is programmed by the AltoStudio using a number of pre-selected logic functions which can be applied to selected items in the DataInstance and DIList. Alternatively, a window is provided to add a Java programming capability to meet specific requirements of the application.

In principle, any mathematical or logical operation may be performed on any of the items in the data returned from the data base. QueryActionLogic provides in-stream processing of data removed from operations in the data base. Operations can be performed on data away from the administered data base environment.

The results of the query are then returned from QuerySession via the servlets to the client application.

---

### *EJB SuperClass*

AltoServer is constructed out of Enterprise Java Beans (EJBs).

All AltoWeb EJBs are derived (extended) from the EJB parent superclass which, in turn, controls all locking levels (password access) to the server.

There are two level of access

- No lock
- Exclusive lock access

The immediate children of the superclass are the Developer and Stateless classes.



## **Projects**

Each project is identified by two strings:

- A group, and
- A name

The group file is of the form.

a/b/cc (EQ 1)

The name file is identified by a letter.

d (EQ 2)

The root directory is of the form.

a/b/cc/d (EQ 3)

AltoStudio creates a Project during a stateless DeveloperSession with AltoServer and consists of three types of classes.

- **DataProject**

The Data is the MetaObject and ObjectConnection definitions and the associated functional requirements such as data source connection information and logic model. The DataProject class provides objects for the MetaCluster and subsequently for the entity beans during QuerySessions.

- **LayoutProject**

The Layout is the programmable functional characteristics of the client. This includes facet, band and popup functionality. The LayoutProject class embodies the layout characteristics defined by AltoStudio.

- **TemplateProject**

The Template is the unchanging visual and functional representation used by a client and is created exclusively by AltoStudio. For example, the division of the screen into three parts – structure, context and structure is a Template feature. The TemplateProject class has no runtime instance.

Data is the MetaObject and ObjectConnection definitions. This includes associated functional requirements such as data source connection information and logic model. DataProject class provides objects for the MetaCluster and subsequently for the entity beans during QuerySessions.

---

## *Projects*

Projects, created and saved using AltoStudio, provide a summary of the components and values. The purpose of a Project is to run queries and obtain results (data instances).

Project performs data access and processing to obtain a result for a particular goal. It is defined by a TimeStamp, which is a reference time which makes the project unique. The TimeStamp is the start point for data access and processing and determines the unique result of queries.

A project consists of a MetaModel and a DataModel.

1. MetaModel is the abstract representation of data which defines the "business" objective for data access and processing. MetaModel consists of MetaObjects, ObjectConnections, and LogicObjects (QueryLogicAction).
2. DataModel is the abstract mapping of data to actual (external) data sources to obtain a result for a data access and processing objective. The DataModel consists of DataSet objects, Field objects, DataSources and their connections.

Project takes business logic and requests and translates them into actual results from data sources.

The programs which describe multiple Projects reside in the AltoServer MetaCluster. They were all created by AltoStudio to be used by AltoServer. MetaObjects are accessed by the server on a client/application request. An entity bean holds the program for a specific Project (consisting of MetaObjects and DataObjects) which was called by the QuerySession.

---

## *ObjectConnections*

An ObjectConnection defines a directional connection between one MetaObject and other MetaObjects. AltoStudio creates ObjectConnections using the Relationship Diagram.

### **Parent and Child Objects**

A parent MetaObject is a MetaObject which already exists.

A child MetaObject is created from the parent by changing the business information or business logic within the parent.

An ObjectConnection may be created between the parent MetaObject and the child MetaObject such that when the parent MetaObject is selected, the child MetaObject results from the parent MetaObject request.

A parent MetaObject is changed to a child based on the query response.

Parent and child MetaObjects usually hold similar information in common from which the child was constructed. This might be a column such as a Primary Key. But they need not necessarily hold information in common. Parent and child MetaObjects often access completely different DataSets.

The child MetaObject returns information that is in some way related to the parent.

---

## *DataSet, Field and DataSource Objects*

DataSet Objects are categories of a multiple element relationship in a database or data source.

DataSet properties are not defined by default. They must be set.

The DataSet interface is used to manage fields and connect to a data source.

## Field Objects

Field Objects are categories of individual items from a Field and can be specified using one of two data type genres.

1. Java types

int, string constants representing the standard Java data types such as int, short, long, float, etc.

2. SQL types

native SQL data types which use integer data as specified in `java.sql.types`

A field in the DataSet is specified as the Primary Key. This is the key field or reference field and is important when joining or manipulating fields and datasets.

## DataSource Objects

DataSource Identifiers are used to specify the columns of a table requested from a data source. The `nativeIdentifier` is the name given to the data source file (sometimes abbreviated `ds`, for reference). For JDBC, it is the table name.

The following table relates DataSource, DataSet and Field relationships.

	Conventional	Web
DataSource	ds (JDBC, User/Password URL)	Directory
DataSet	Table	File, Directory
Field	Column (Pointer)	HTML Parser, XML Parser

The DataSource interface manages connections to the data source and their connections to DataSets identified by DataSetIdentifiers. It manages the introspection feature and returns connection information to and from DataSets.

AltoStudio manages the DataSource interface, and is responsible for configuring and accessing data from the data source. At run time, the DataSource interface creates and services the actual connection.

---

### *Compound MetaObjects and Joins*

---

A Compound MetaObject is a class which refers to more than one DataSource. The fields from the parent MetaObject are the master DataSet. Queries can be executed from one top level DataSet. All the DataSets must come from the same DataSource.

A Join creates multiple MetaObjects from one DataSet.

An InnerJoin creates a MetaObject which is wholly contained within the DataSet. An Outriden brings data fields from more than one DataSet of which one of the DataSets is termed the Master DataSet.

Multiplexing involves the combination of multiple databases to create a MetaObject using multiple connections to create a new database and DataSet. Similarly, with MultipleJoins, there must be a Master DataSet for reference. Clearly, different combinational methods can be used to join DataSets to create MetaObjects.

SlaveJoins is a mapping function that contains DataSources. SlaveJoins represent the masters as keys and JoinField objects as values and is used inside the Compound MetaObject. The class holds two fields from different DataSets that are used for the join. The two DataSets must be from the same DataSource.

AltoStudio permits creation and mapping of Compound MetaObjects and Joins. The classes which manage these features are in the CompoundMetaObjects and JoinField classes. Joins permit data base operations remote from the data source.

---

## XML

Some data sources may be queried with XML as the input and the results may be returned in XML format.

- The xml1 servlet returns the types of values as attributes in a Document Type Description (DTD).
- The xml2 servlet just returns the results as XML without the DTD concurrence.

To access XML files from an outside client, the URL is specified in this property. The property contains a URL that is the prefix of the generated DTDs and will be accessible to the XML users.

The web server should be configured to send requests with this prefix to the directory of the previous property. Or, the "files" property should point to a web server directory with the name in the "publishPath" property.

AltoWeb supports the UTF-8, Cpl252 and US-ASCII character set encoding conventions for XML.

The xml1 servlet returns XML that contains the type of every value as an attribute. The values can be specified in an external DTD (the default), an internal DTD, or not specified at all. You may select from the three values, External, Internal or None.

XML servlets support caching.

## XML Implementation

BaseQuery is an abstract class which extends the HttpServlet class. It is the base for servlets which need access to the QuerySession. It sets the JNDI.

The xmlQuery1 class extends BaseQuery. This servlet returns results in XML format to be compatible with a DTD, whether specified external, internal or not at all. It initializes the servlet configuration, connects to the cache, defines project, MetaObject, field, value, PK and QueryName, as appropriate. It obtains the output stream, provides encoding and finds the DTD.

The xmlQuery2 class extends BaseQuery. This servlet returns results in XML format without a DTD. It queries the HTTP servlet, finds encoding, defines the project data, and processes the query providing an output stream.

The DTD Generator class is part of the DeveloperSession and is set using AltoStudio. It gets the server properties, project xml files and publish path. It obtains the DateTime instance, the appropriate Java data types for the DTD parameters and MetaObjects, and checks for a data source. The DTD Generator is part of the AltoStudio application development process.

The Project Container (beans. project.dataProject) contains all the project data including xml support.

The XMLDataIdentifier gets or sets the XML data type. These are defined from the Document Object Model (DOM) w3c standards at org.w3c.dom.node.

NODE	CODE	NODE	CODE
ELEMENT_NODE	1	PROCESSING_INSTRUCTION_NODE	7
ATTRIBUTE_NODE	2	COMMENT_NODE	8
TEXT_NODE	3	DOCUMENT_NODE	9
CDATA_SECTION_NODE	4	DOCUMENT_TYPE_NODE	10
ENTITY_REFERENCE	5	DOCUMENT_FRAGMENT_NODE	11
ENTITY_NODE	6	NOTATION_NODE	12

The `XMLDataSource` class extends the `DataSourceImpl` class. It performs the same kind of functions as the `DataSource` class. It gets and sets the data source (the URL or file identifier) and provides introspection of all `DataSet` and `Field` Identifiers and connections. In addition, it sets the `XMLParser` to null, and defines the `SAXParser` (or other) to process the incoming xml data stream.

The `xmlDynamicQuery` extends the `DynamicQueryImpl` class. This class runs the query on the XML data source. Like the `DynamicQuery`, the query is defined by `MetaObject`, map conditions, properties, `DIList` and max number of lines in the result. It runs the `Query Filter` and `Parser` on the result.

The `xmlWriter` class in the `meta.dataInstance` package returns results in xml format as either a `DataInstance` or a `DIList`.

The classes in the `data.xml.parser` package are concerned with identifying xml queries and results and making sure that the xml data is well-formed and meets DTD and validation criteria of other parsers.

The classes in the `data.xml.tree` package process the xml data. The `TreeWalker` class starts at the root node and advances through the xml file to determine what action to take.



The ElementNode, AttributeNode, TreeNode and TextNode classes provide information about how to process the data.

---

### *MetaObject Creation and Mapping*

AltoStudio allows the user to create and map MetaObjects by the following procedures:

1. Create or open a project.
2. Establish a connection to the raw data.
  - Select data source
  - Enter connection string
  - Enter user login/password
3. Select DataSets and Fields to form the (new) MetaObject(s).
  - Save MetaObject in Project.
  - Select and save desired MetaObjects.
4. Draw connections between MetaObjects.
5. Define connections.
6. Save ObjectConnections as part of Project.

---

### *Custom Logic Creation and Editing*

AltoStudio allows the user to create and edit logic by the following procedure using the ProgrammableJavaLogic classes.

1. create a File
2. beforeAll
3. afterIteration
4. afterAll

1. *Staphylococcus aureus* (Staph. aureus)  
 2. *Staphylococcus epidermidis* (Staph. epidermidis)  
 3. *Staphylococcus saprophyticus* (Staph. saprophyticus)  
 4. *Staphylococcus carnosus* (Staph. carnosus)  
 5. *Staphylococcus sciuri* (Staph. sciuri)  
 6. *Staphylococcus hyacinthi* (Staph. hyacinthi)  
 7. *Staphylococcus albus* (Staph. albus)  
 8. *Staphylococcus citreus* (Staph. citreus)  
 9. *Staphylococcus gelae* (Staph. gelae)  
 10. *Staphylococcus lentus* (Staph. lentus)  
 11. *Staphylococcus maritimus* (Staph. maritimus)  
 12. *Staphylococcus pasteurii* (Staph. pasteurii)  
 13. *Staphylococcus schweinfurthii* (Staph. schweinfurthii)  
 14. *Staphylococcus simulans* (Staph. simulans)  
 15. *Staphylococcus warneri* (Staph. warneri)

---

The following figures illustrate the relationships discussed in Chapter 4 "Theory of Operation" on page 31.

It includes the following:

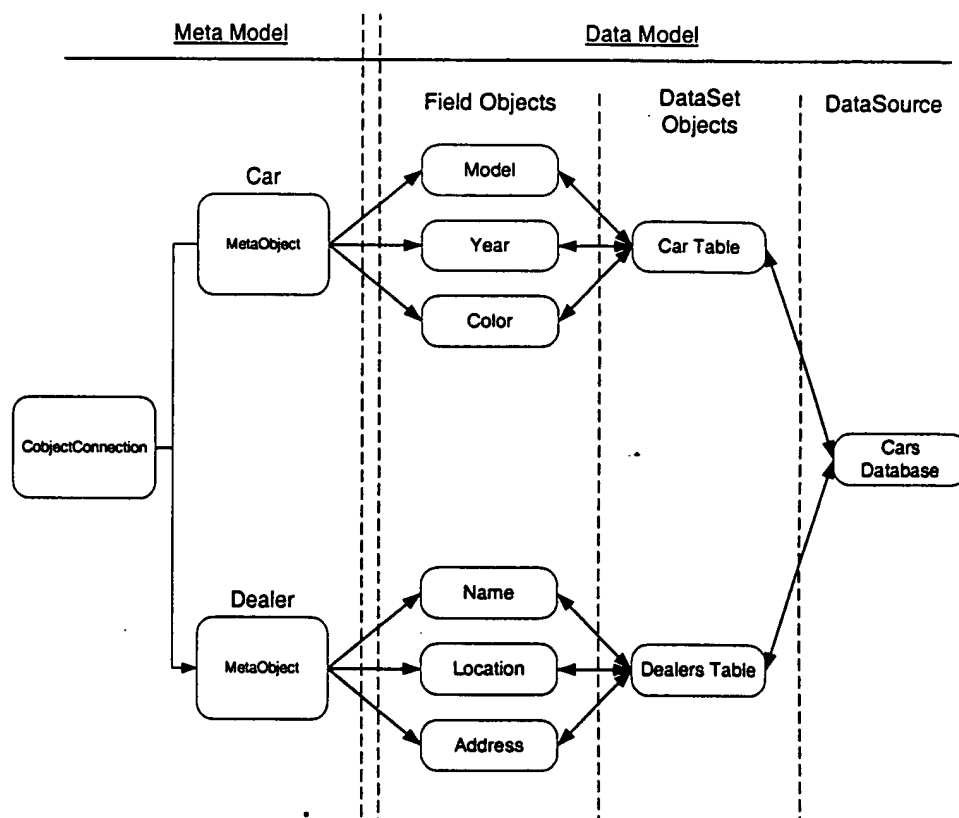
1. "The Structure of Classes in a Project" on page 52
2. "The Content of a Project – Data" on page 53
3. "Implementation of DataObject Interfaces for RDBMS" on page 54
4. "Data Mapping in the Application" on page 55
5. "Flow of Query Execution" on page 56
6. "Flow of Execution" on page 58
7. "Flow of Client Request to the Server" on page 59
8. "AltoServer Application" on page 60



---

*The Content of a Project – Data*

---

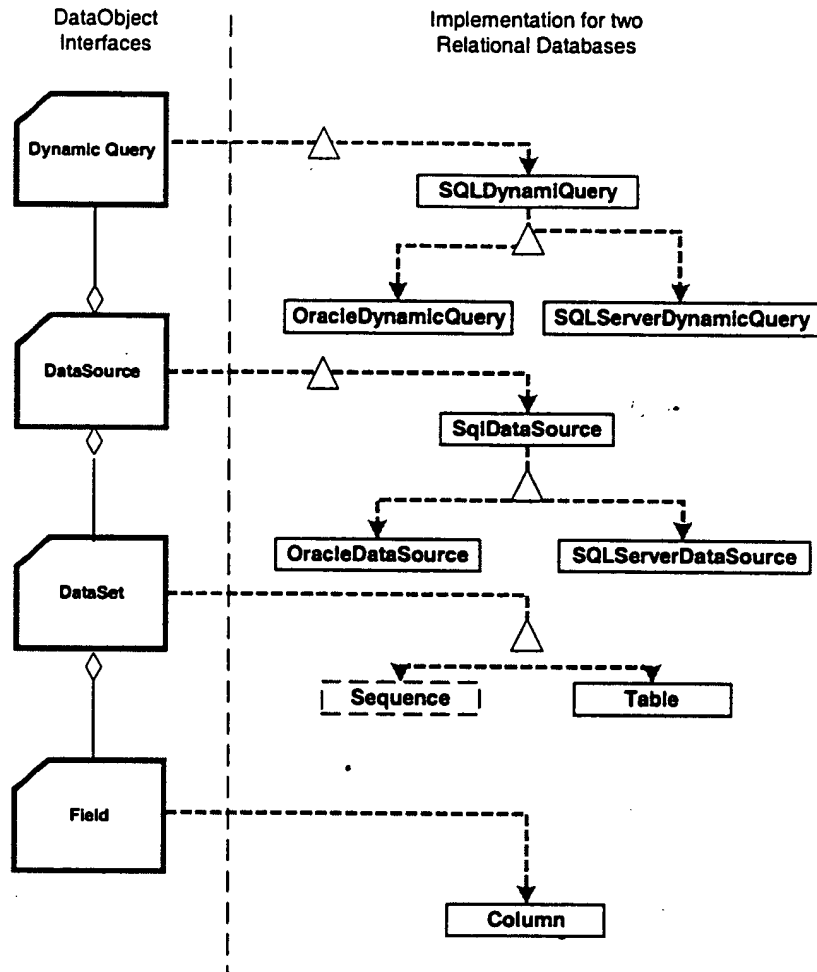


Content of a Project illustrates the relationship between MetaObjects, Field Objects, DataSet objects and a DataSource.

Field Objects represent one item or column and forms part of a DataSet or table. The DataSource is the actual place to look for the data.

In this Project, there are two MetaObjects – Car and Dealer. It is important to specify the Project timestamp, since the database will be different at different times. This project specifies to look for data about Car and Dealer in a Cars Database.

## Implementation of DataObject Interfaces for RDBMS

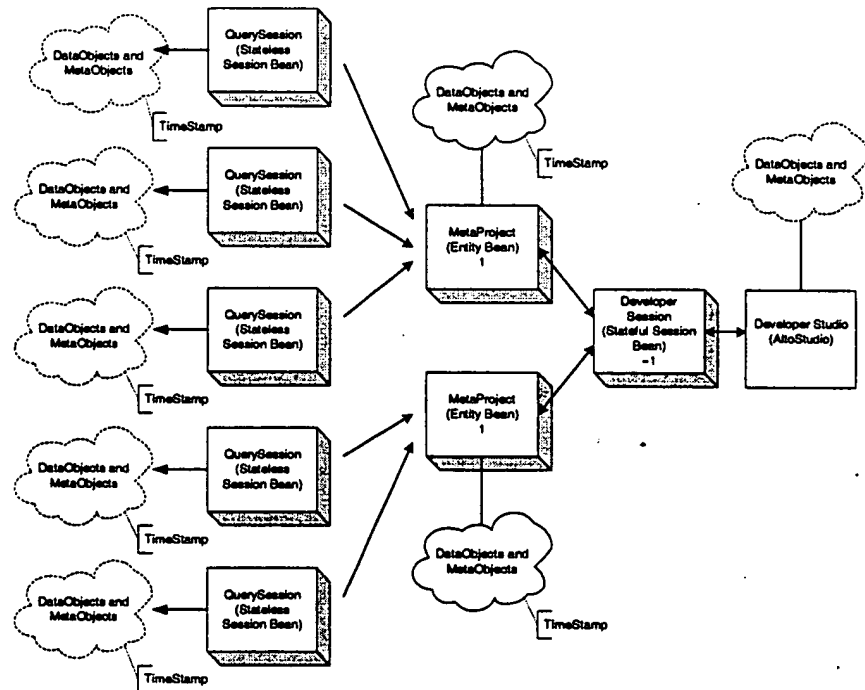


Implementation of DataObject Interfaces for a Relational Database Management System shows the relationship between Field, DataSet, DataSource and DynamicQuery for Oracle and SQL Server RDBMSs.

---

## Data Mapping in the Application

---



### Data Mapping in the Application

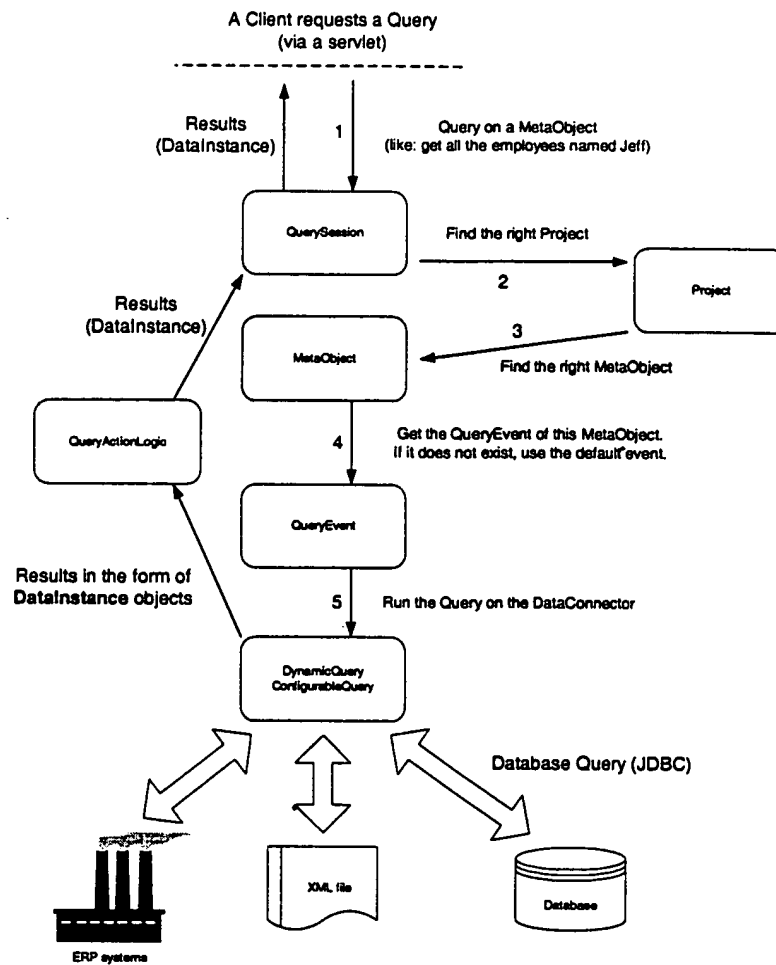
There is only one original Project file for each Project (MetaObjects and DataObjects), which is stored in the Entity Bean. Each instance of the Project is cloned and validated by reference to its TimeStamp. Each clone is uniquely defined by its TimeStamp.

Data Mapping shows the flow of data through the application.

MetaObjects and DataObjects are created in AltoStudio by the DeveloperSession bean. The Project (MetaObject and DataObject) is defined in the Entity bean.

The original copy of the Project is validated at every use by comparing the time stamp. The session bean sets up a stateless session to the actual data sources.

## Flow of Query Execution



Flow of Query Execution shows the process of execution from the time that a client executes a query (the servlet) to the time that the client receives results (a DataInstance).

During the QuerySession, the object finds the requested Project which, in turn, finds the MetaObject.

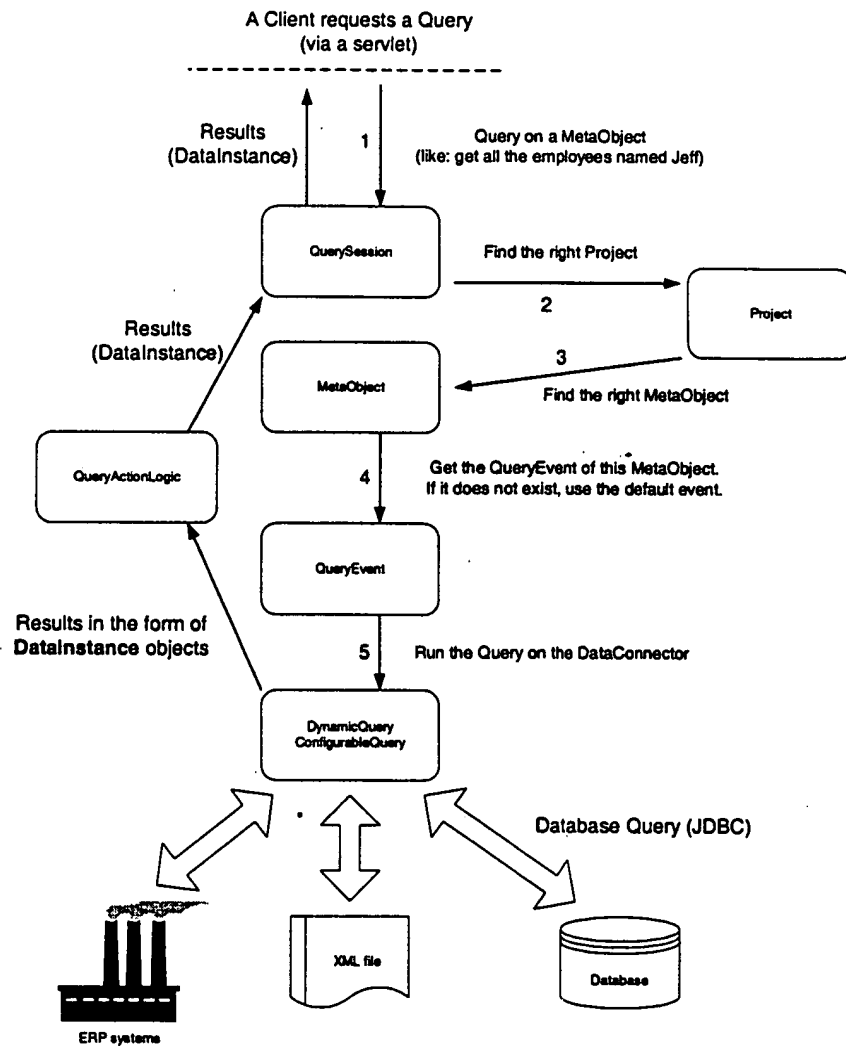


The response from the database is a QueryEvent i.e., what to look for in the data file. Queries to data files can be either a Dynamic-Query or a ConfigurableQuery.

A ConfigurableQuery limits the range of requests to part of the database by various methods.

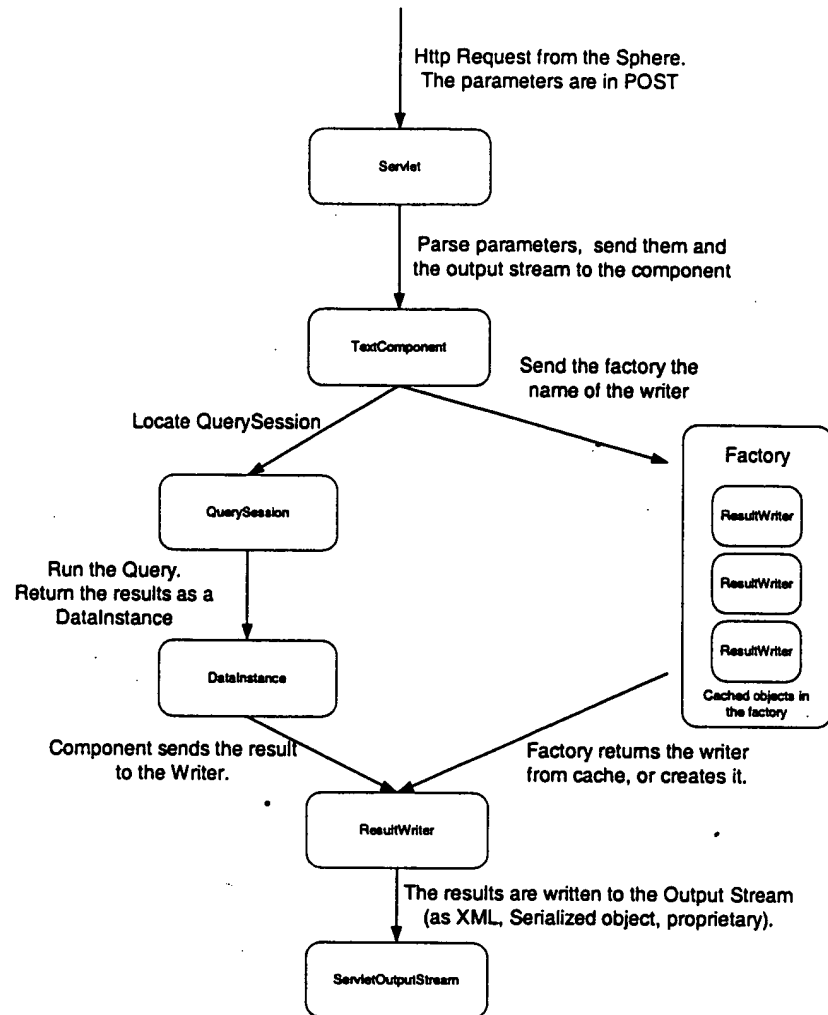
A Dynamic Query accesses the unrestricted database. The result is a DataInstance object. QueryActionLogic decides what to do with the requested data by applying a logic model which processes the DataInstance result mathematically or logically. The results are returned to the client through the servlet interface.

## Flow of Execution



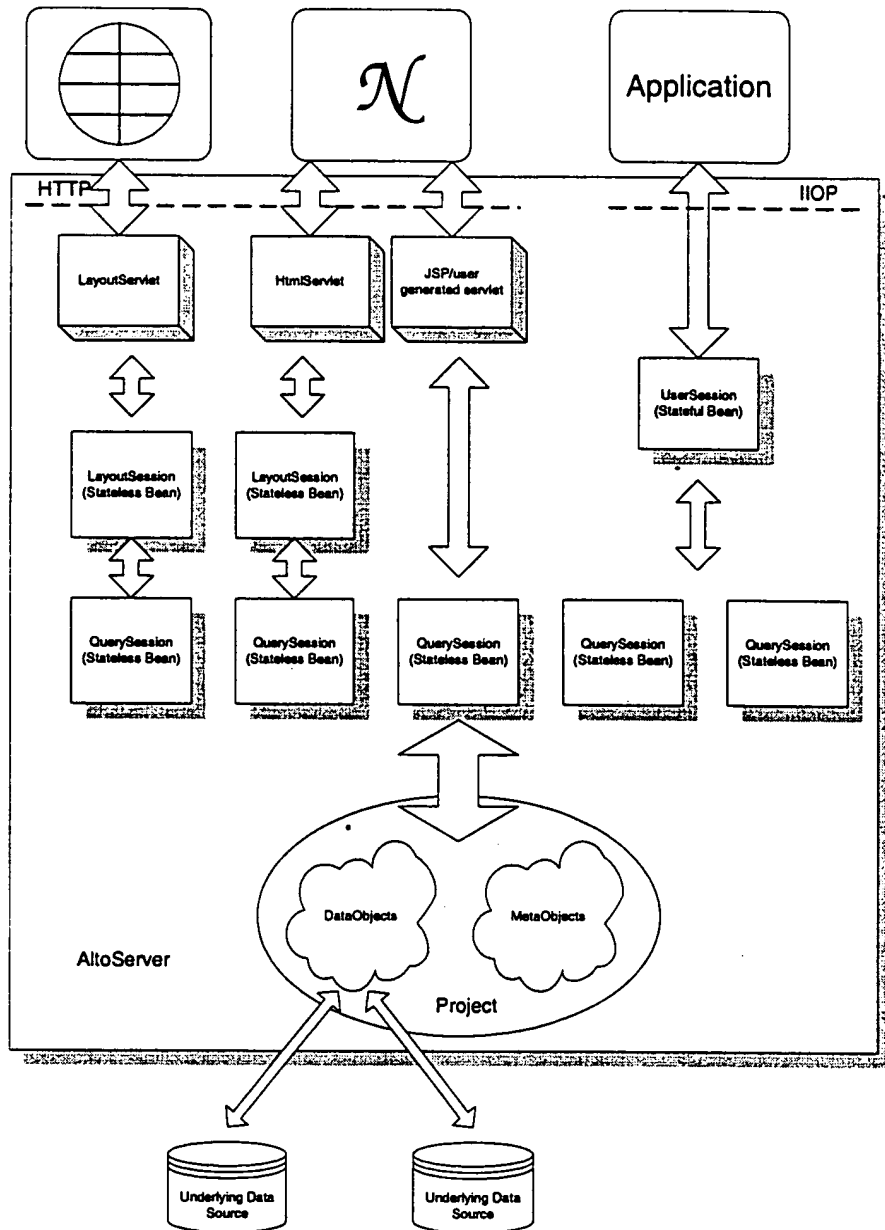
Flow of Execution shows the execution flow versus object ownership for the processing of a query.

## Flow of Client Request to the Server



Flow of Client Request to the Server shows the sequence of events beginning with an HTTP request from a client to the realization of a response in the output stream to the client.

## AltoServer Application



The figure, AltoWeb Server, is an example of how the AltoServer processes a request for data from an underlying data source.

Copyright © 1999 by AltaServer

AltoServer is a J2EE application that uses Enterprise Java Beans (EJBs).

### *AltoServer Beans Super Package*

The AltoServer beans super package contains AltoServer EJBs.

TABLE 1. AltoServer Beans Super Package

Package	Purpose
beans	
beans.counter	Manages persistence and transaction management including login and password in the client.
beans.developer	Manages the DeveloperSession with AltoStudio.
beans.developer.data	Sets up the DataObjects with AltoStudio.
beans.developer.layout	Sets up the layout with AltoStudio.
beans.developer.template	Sets up the template with AltoStudio.
beans.layoutSession	Manages the layout features in runtime (client).

**TABLE 1. AltoServer Beans Super Package (Continued)**

<b>Package</b>	<b>Purpose</b>
beans.layoutSession2	Manages the layout features in runtime (client)
beans.project	Manages the Project in runtime (client).
beans.project.dataProject	Manages the data processing and properties in runtime (client).
beans.project.layoutProject	Manages the layout properties in runtime (client).
beans.project.templateProject	Manages the template properties in runtime (client).
beans.querySession	Manages the querying process in runtime (client).



## Altoserver Application Super Package

TABLE 2. AltoServer Application Super Package

Package	Purpose
cache	Manages timing, synchronization, threading and cache utilization.
clientAccess	Manages client access interfaces.
clientAccess.jsp	Manages client Java Server Pages (client page access).
clientAccess.servlets	Manages client Servlets.
data	Manages Data Objects and Data Source connections
data.file	Provides and manages File Access queries.
data.sql	Provides and manages SQL queries.
data.general	Provides and manages Programmable Queries from AltoStudio.
data.web	Provides and manages Web queries.
data.web.htmlParser	Converts and manages links to HTML tags.
data.web.tree	Provides web nodes to provide query execution.
data.xml	Provides and manages XML queries
data.xml.parser	Translates and validates XML inputs and outputs.
data.xml.tree	Advances through XML nodes to perform query execution.
layout.ProjectData	Manages Layout interface for client.
.meta	Defines and manages MetaObjects, Object Connections, Logic Model and Query Definition.
meta.dataInstance	Defines and manages results from data bases.
meta.logic	Provides and manages the capability to perform logic operations in Java on results from data base.
studioData	Provides AltoStudio font, color, dimensions, etc., features.
templateProjectData	Manages template interface for client.
util	Provides a variety of utilities including JNDI naming convention, server properties, and filters.

---

## *AltoServer EJB Implementation*

---

EJBs run within the EJB container. The EJB container controls the beans and provides them with system level services.

The EJB Container provides the following services.

- Transaction management
- Security
- Remote client connectivity
- Bean life cycle management
- Data base connection pooling

Each EJB is implemented as an EJB class, a remote interface, and a home interface. They are then saved as a .jar file.

Where appropriate, the EJB class is supported by Helper classes. A Deployment Descriptor is used to facilitate installation.

Data base connections are managed by Application classes supported by a Java Naming and Directory Interface (JNDI) which is used as a lookup table. The JNDI name links to the URL or file name of the data base.

The implementation has bean managed persistence. The entity beans contain the calls that access the data bases.

JSP implementation involves use of Java bean as a proxy to access enterprise beans. It is coded within an HTML code segment.

Servlets retrieve user data, looks up the session bean, and passes the data to the session bean. Upon receiving a value back from the session bean, it creates an HTML page to display the returned value or sends the value separately the client program.

The client can be a Java-RMI connection, or another language plus CORBA-IIOP connection.

---

## *WebLogic Properties File*

This file should be appended to your weblogic.properties file.

It defines the setup specific to AltoWeb. Some of these properties override setup configurations documented earlier in this file.

### **User Generated Entries**

weblogic.httpd.servlet.reloadCheckSecs=2

weblogic.httpd.register.ejbTest=clientAccess.servlets.TestQueryServlet

weblogic.httpd.register.stateServlet=clientAccess.servlets.LayoutServlet

weblogic.httpd.register.htmlView=clientAccess.servlets.HtmlView

weblogic.httpd.register.query=qa.http.HttpQueryExecution

weblogic.httpd.register.qv=qa.http.QueryViewer

weblogic.httpd.register.xml1=clientAccess.servlets.XmlQuery1

weblogic.httpd.register.xml2=clientAccess.servlets.XmlQuery2

weblogic.ejb.deploy=\

d:/temp/classes/devDataProject.jar, \

d:/temp/classes/devLayoutProject.jar, \

d:/temp/classes/devTemplateProject.jar, \

d:/temp/classes/querySession.jar, \

d:/temp/classes/layoutSession2.jar, \

d:/temp/classes/counterBean.jar

weblogic.httpd.documentRoot=d:/temp/httpd/html

### **Connection Pool for Testing Performance of Beans**

weblogic.jdbc.connectionPool.test=\

weblogic.jdbc.connectionPool.otest=\

url=jdbc:weblogic:oracle,\

driver=weblogic.jdbc.oci.Driver,\

loginDelaySecs=1,\

initialCapacity=0,\

maxCapacity=30,\

capacityIncrement=2,\

allowShrinking=true,\

shrinkPeriodMins=15,\

refreshMinutes=10,\

testTable=dual,\

props=user=scott;password=tiger;server=oraserver1

props=user=scott;password=tiger;server=oraserver1

weblogic.allow.reserve.weblogic.jdbc.connectionPool.otest=guest

weblogic.allow.reserve.weblogic.jdbc.connectionPool.test=guest

weblogic.password.testUser=testtest  
weblogic.password.testUser1=testtest  
weblogic.password.testUser2=testtest  
weblogic.password.testUser3=testtest

---

### *AltoWeb Server Properties*

As these properties are read by AltoServer when loading, changing them requires server restart.

Place this file in the directory where the server starts. *These are not application server properties*: they are kept separately.

When specifying file and directory names, use your operating system file separator. When a property contains a list of properties, add a line for each entry.

### **Projects Data Storage and Update**

Projects data will be stored in the following directory.

project.persistenceRoot=d:\temp\ejbStorage (EQ 4)

The update interval for loading data projects (seconds) is defined by the following.

project.DataProject.update=6000 (EQ 5)

## XML Output

Some projects can be queried and the results obtained in XML format.

The "xml1" servlet returns the types of the values as attributes in a DTD. The results are then stored in this directory:

`project.xml.files=d:/temp/httpd/html/xml` (EQ 6)

To access the files from the outside, the URL must be specified in this property. This property will contain a URL that is the prefix of the generated DTD's and should be accessible to the XML users.

The web server should be configured to send requests with this prefix to the directory in the previous property. Alternatively, the "files" property should point to a web server directory with the name in the "publishPath" property

`project.xml.publishPath=http://localhost:7001/xml` (EQ 7)

`project.xml.publishPath=/xml` (EQ 8)

Three encodings of XML output characters are supported:

1. UTF-8
2. Cp1252
3. us-ascii

`project.xml.encoding=UTF-8` (EQ 9)

The "xml1" servlet returns XML that contains the type of every value as an attribute. The values can be specified in an external DTD (the default), internal DTD, or not specified at all.

There are three values:

1. internal
2. external
3. none

`project.xml.xml1.dtd=external` (EQ 10)

XML servlets support caching. Here, we define both "xml1" and "xml2" servlets to share the same cache named "xmlCache."

project.xml.xml1.cache=xmlCache (EQ 11)

project.xml.xml2.cache=xmlCache (EQ 12)

## Java Code Generation

Code generation for templates: temp directory for generating and compiling java code

layout.codeGeneration.temp=d:\temp\codeGeneration (EQ 13)

The resulting classes are copied to the web server directory so they could be loaded by applets.

This directory should be the classes directory that the applets are loaded from.

layout.codeGeneration.output=d:\temp\httpd\html\AltoClient\classes (EQ 14)

## Compile command

The server will use this compilation command.

%temp% will be replaced with the temp directory (EQ 15)

%files% will be replaced with the java class file names (EQ 16)

%classpath% will be replaced with classpath of the server (EQ 17)

This parameter is optional.

layout.codeGeneration.command=javac -d %temp% -classpath  
d:\temp\httpd\html\AltoClient\classes;d:\jdk1.1.7\lib\classes.zip%files% (EQ 18)

## Code Generation for Logic

These are directories for java logic. *They SHOULD NOT be in the classpath of the WebLogic server or client.*

logic.userGenerated.source=d:\temp\userLogic\src (EQ 19)

logic.userGenerated.output=d:\temp\userLogic\classes (EQ 20)

logic.userGenerated.temp=d:\temp\userLogic\classes\tmp (EQ 21)

This is the compile command, the output directory needs to be as output above.

need the -d outputdir to generate the package structure (EQ 22)

%files% will be replaced with the java code file names. (EQ 23)

logic.userGenerated.command=javaegd:\temp\userLogic\classes\tmp%files%(EQ 24)

## HTML Generation

The web server should point to the output directory for generated html files.

layout.html.output=d:\temp\httpd\html (EQ 25)

## Security

The system login and password for AltoStudio login should only be changed by administrators.

security.admin.login=foo (EQ 26)

security.admin.password=bar (EQ 27)



## Accessible Directories In Your File System

AltoServer will not allow access to a file or a directory that is not in this list.

security.filesystem.accessible=d:\temp (EQ 28)

security.filesystem.accessible=d:\httpd\html\public (EQ 29)

## Restricted Domains

The server will NOT allow access to any domain that starts with one of these. Make this list as short as possible, since it has performance overhead.

localhost is always forbidden. (EQ 30)

security.http.forbidden=www.virus.net/hostile (EQ 31)

security.http.forbidden=www.whitehouse.com (EQ 32)

security.http.forbidden=test2 (EQ 33)

## Data Sources

### The List of Available Data Sources

If another data connectivity package is added, or purchased separately, ensure that the classes are in the server classpath. Then, look in the documentation to find which name you have to enter here.

project.DataProject.dataSources.available=data.sql.SqlDataSource (EQ 34)

project.DataProject.dataSources.available=data.web.WebDataSource (EQ 35)

project.DataProject.dataSources.available=data.file.FileDataSource (EQ 36)

project.DataProject.dataSources.available=data.xml.XmlDataSource (EQ 37)

## A Limit on the Number of Results in a Query

Some data sources set their own limits that override this one.

The limit protects against hostile queries, and should be set accordingly.

`dataSources.maxLines=1000` (EQ 38)

## DataSource Specific Limits on Query Result Size

`dataSources.maxLines.sql=20000` (EQ 39)

`dataSources.maxLines.file=200` (EQ 40)

## Logic Classes

This property contains the logic classes that are available.

ProgrammableJavaLogic will show all the classes that were generated by the user.

`project.DataProject.logic.available=meta.logic.TextTemplateLogic` (EQ 41)

`project.DataProject.logic.available=meta.logic.ProgrammableJavaLogic` (EQ 42)

## Cache Management

Every cache in the server can be configured using the following parameters.

HPM — Average hits per minute rate that qualifies a page to get into the cache

cacheSize — Average size in bytes (of the whole cache)

maxFileSize — To prevent huge files from blocking the cache, files over this size will not get in

updateSec — After this time interval, the cache is refreshed and cleaned

It is important to notice that files get in and out of the cache all the time. This is an internal statics update.

A cache can contain *ANY* binary data. All values have reasonable defaults, but it is still recommended that the server developer will test different configurations.

The cache is given a name and can be shared by many components that support caching. (Many servlets and beans can share the same cache).

In the current configuration, the cache "xmlCache" is shared by both xml1 and xml2 servlets.

XML servlets cache:

cache.xmlCache.HPM=41M size cache (EQ 43)

cache.xmlCache.cacheSize=1000000 (EQ 44)

100K maximum page size (EQ 45)

cache.xmlCache.maxFileSize=100000 (EQ 46)

cache.xmlCache.updateSec=60 (EQ 47)

Year	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	

---

The following interface classes are described in this chapter.

- "QuerySession" on page 80
- "QueryEvent" on page 91
- "DataInstance" on page 93
- "DIList" on page 99
- "DataSource" on page 100
- "DataSet" on page 114
- "Field" on page 120
- "BasicQuery" on page 124
- "DynamicQuery" on page 126
- "ConfigurableQuery" on page 132
- "QueryLogicAction" on page 146

QueryEvent is not an interface class but an abstract class and that QuerySession is an EJB implementation with home and remote interfaces, helper classes, and a deployment descriptor.

DataSource, DataSet and Field have implemented classes associated with them.

## Parameter Descriptions

The following table summarizes parameters used by the AltoServer API.

**TABLE 3. Parameter Descriptions**

Parameter	Description	Allowable Data Type/ Values
Project	Project Name	String
MetaObject	A MetaObject	String
Meta	A MetaObject	String
MyMeta	A Master MetaObject	String
OtherMeta	A MetaObject to be Joined to a Master MetaObject	String
parent	A MetaObject	String
children	A MetaObject derived from a Parent	String
upperBoundMetaObject	A MetaObject whose numeric values are upper bounded	String
lowerBoundMetaObject	A MetaObject whose numeric values are lower bounded	String
dataset(s)	DataSet (s)	String
datasetIdentifier	Name of a Dataset	String
datasetIdentifiers	Name of more than one DataSet	String
field	Field	String
myField	A Master Field	String
otherField	A field to be Joined to a Master Field	String
fieldIdentifier	Name of a Field	String
primary key	Reference Field in a DataSet	String or Value
pk	Primary Key	String or Value

## Parameter Descriptions

**TABLE 3. Parameter Descriptions (Continued)**

Parameter	Description	Allowable Data Type/ Values
upperBoundField	Name of a Field	String
lowerBoundField	Name of a Field	String
orderField		
connection(s)	DataSources which are connected	String
connectables	DataSources which can be connected to	String
connectionProps	URL, Login, Password	String
source	A DataSource designator	String or Value
nativeIdentifier	A DataSource designator used to achieve or request a connection	String or Value
property	An item in a data base	String
selectProperties	An item in a data base	String
propName	An item in a data base	String
value		Value
conditions	A map	Value
list	A list of items	String(s)
map	A list of items	String
upperBound	Upper limit	Numeric
lowerBound	Lower limit	Numeric
upperBoundParameter	Upper limit	Numeric
lowerBoundParameter	Lower limit	Numeric
parameter(s)	An item in a data base	String or Value
parentParameters	An item in a data base	String or Value
parameterNames	An item in a database	String or Value
logic	An equation	String
name	An item in a data base	String

TABLE 3. Parameter Descriptions (Continued)

Parameter	Description	Allowable Data Type/Values
queryName	Query	String
customQuery	Custom Query	
queryData	DataInstance or DIList	

## *QuerySession*

package beans.querySession

(EQ 48)

### Description

QuerySession class provides AltoServer interface and functionality management. It handles the runtime execution of requests, queries and results.

It manages the session, i.e., the association between the two communicating end-points. The connection is maintained between client and data source while the two points are communicating and the mid-tier functionality is managed and executed.

QuerySession is a program which resides on the session beans. It is a stateless session. (It does not keep information.)

QuerySession requests and receives data based on criteria for the query. These include primary key reference, matching of items, range conditions, specified query, MetaObject name, property values, ObjectConnection and parent/children of MetaObjects or DataInstances.

QuerySession returns either a DataInstance or a DIList.



## Methods

getSingleByPK

(EQ 49)

public DataInstance getSingleByPK(String project, String metaObject, String[] selectProperties, Object pk) throws ConnectionException, BadQueryException, RemoteException

### *Description*

Invokes a query if the meta object is associated with a primary key.

### *Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

pk - a field in the data base identified as the Primary Key (any data type, usually a string)

### *Return Value*

Returns one data instance if it is part of the primary key. All the parameters must be true.

### *Exceptions*

ConnectionException, BadQueryException, RemoteException

getSingleByMatch

(EQ 50)

public DataInstance getSingleByMatch(String project, String metaObject, String[] selectProperties,

String property, Object value) throws ConnectionException, BadQueryException, RemoteException

*Description*

Invokes the first query from the list.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

property - an item in the database (any data type, usually a string)

value - an item in the data base (any data type)

*Return Value*

Returns one data instance. It identifies and returns the first match only. All parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, RemoteException

`getSingleByMatch`

(EQ 51)

```
public DataInstance getSingleByMatch(String project, String
metaObject, String[] selectProperties, String property,
List values)throws ConnectionException, BadQueryException,
RemoteException
```

*Description*

Obtains a DataInstance by a query on more than one field.

### *Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

property - an item in the database (any data type, usually a string)

values - an list of items in the data base (any data type)

### *Return Value*

Returns one data instance or DI List. It identifies and returns the first match only from the list. The search is on more than one field. All the parameters must be true.

### *Exceptions*

ConnectionException, BadQueryException, RemoteException

getSingleByRange

(EQ 52)

```
public DataInstance getSingleByRange(String project, String  
metaObject, String[] selectProperties,
```

```
String property, Object upperBound, Object lowerBound)throws  
ConnectionException,
```

```
BadQueryException, RemoteException
```

### *Description*

Obtains a DataInstance by comparing the data base against properties and within a range of values.

### *Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

property - an item in the database (any data type, usually a string)

upperBound - a numeric data type (e.g. int, double, short, float, etc.)

lowerBound - a numeric data type (e.g. int, double, short, float, etc.)

### *Return Value*

Returns one data instance if it is within a range of numeric parameters specified by upper and lower bound values. All the parameters must be true.

### *Exceptions*

ConnectionException, BadQueryException, RemoteException

getSingleInvokeQuery

(EQ 53)

```
public DataInstance getSingleInvokeQuery(String project, String
metaObject, String[] selectProperties, String queryName, Map
values) throws ConnectionException, BadQueryException, Remo-
teException
```

*Description*

Invokes a query on a meta object by query name and parameters.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

queryName - a specified query with predetermined criteria (a string)

values - numeric data types which specify a map of the part of the table in which to look

*Return Value*

Returns one data instance by query and parameters. All parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, RemoteException

getManyByPK

(EQ 54)

```
public DIList getManyByPK(String project, String metaObject,  
String[] selectProperties, List pks)throws ConnectionException,  
BadQueryException, RemoteException
```

*Description*

Invokes queries on Primary Key fields in the data base.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type,  
usually a string)

pks - fields in the data base identified as Primary Keys (any  
data type, usually a string)

*Return Value*

Returns DI List if it is associated with primary keys. All the  
parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, RemoteException

getManyByMatch

(EQ 55)

```
public DIList getManyByMatch(String project, String metaObject,  
String[] selectProperties, Map conditions)throws ConnectionEx-  
ception, BadQueryException, RemoteException
```

*Description*

Returns a DI List if there is a match. It identifies and returns the as many matches as it finds. All the parameters must be true.

*Parameters*

project - the project name (a string) metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

conditions - numeric data types which specify a map of the part of the table in which to look

*Return Value*

Returns DI List if there is a match and it lies within a map of items in the table. All the parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, Remote-Exception

getManyByParent

(EQ 56)

public DIList getManyByParent(String project, String metaObject, String[] selectProperties, String property, List values, DataInstance parent)throws ConnectionException, BadQueryException, RemoteException

*Description*

Invokes a query on the parent DataInstance.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

property - an item in the data base (any data type, usually a string)

values - numeric data types which specify a map of the part of the table in which to look

parent - the name of the parent data instance (a string)

*Return Value*

Returns a DI List with reference to the parent DataInstance. All the parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, RemoteException



getManyInvokeQuery

(EQ 57)

```
public DIList getManyInvokeQuery(String project, String metaObject, String[] selectProperties, String queryName, Map values)throws ConnectionException, BadQueryException, RemoteException
```

*Description*

Invokes a query on a meta object by query name and parameters.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data type, usually a string)

queryName - a specified query with predetermined criteria (a string)

values - numeric data types which specify a map of the part of the table in which to look

*Return Value*

Returns a DI List based on query name and parameters. All the parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, RemoteException

getChildren

(EQ 58)

```
public DIList getChildren(String project, String metaObject,  
String[] selectProperties, DataInstance parent,
```

```
String connection)throws ConnectionException, BadQueryExcep-  
tion, RemoteException
```

*Description*

Invokes a query on the "children" of a parent DataIn-  
stance.

*Parameters*

project - the project name (a string)

metaObject - the metaObject name (a string)

selectProperties - an item in the database (any data  
type, usually a string)

parent - the name of the parent DataInstance

connection - the name of the data source connection  
(a string)

*Return Value*

Returns DI List if there is a match. This method needs  
an instance of DataInstance from the previous query.  
These methods get instances of other objects that  
are the "children" of this object. These are calculated  
from the ObjectConnection classes of this object. All  
the parameters must be true.

*Exceptions*

ConnectionException, BadQueryException, Remote-  
Exception

reloadModel

(EQ 59)

public void reloadModel() throws RemoteException

*Description*

Causes a refresh of the project data in the bean.

---

**QueryEvent**

package meta

**Description**

The QueryEvent class expedites the query as defined by a MetaObject. A QueryEvent is a request to expedite a query. It can be used to modify the query criteria. During runtime, the system executes requests. When a QueryEvent has occurred, the underlying data source is queried, and it sends a result.

The result is either a DataInstance or a DIList.

The QueryEvent class is associated with the configurableQE class. This is a class that stores a configurable query. This class is stored in the metaObject or ObjectConnection class. If the system cannot find the query, it invokes a default query event at runtime.

The DefaultQueryEvent class provides for anonymous query, when the QuerySession cannot find the query from the MetaObject on the entity bean. It executes a QueryEvent and returns the result.

## Methods

getSingle

(EQ 60)

public abstract DataInstance getSingle(String[] selectedProperties, Map conditions, DataInstance parent, QueryLogicAction logic) throws ConnectionException, BadQueryException

### *Description*

Executes a QueryEvent which returns a DataInstance as result.

### *Parameters*

selectedProperties - an item in the database (any data type, usually a string)

conditions - numeric data types which specify a map of the part of the table in which to look

parent - the name of the parent data instance

logic - the logical operations

### *Return Value*

Returns a DataInstance.

### *Exceptions*

ConnectionException, BadQueryException

getMany

(EQ 61)

public abstract DList getMany(String[] selectedProperties, Map conditions, DataInstance parent, QueryLogicAction logic) throws ConnectionException, BadQueryException

*Description*

Executes a QueryEvent which returns a DIList as result.

*Parameters*

selectedProperties - an item in the database (any data type, usually a string)

conditions - numeric data types which specify a map of the part of the table in which to look

parent - the name of the parent data instance

logic - the logical operations

*Return Value*

Returns a DIList. Returns query.getMany(selectedProperties, conditions, parent, logic).

*Exceptions*

ConnectionException, BadQueryException

---

**DataInstance**

package meta.dataInstance

**Description**

A result can have one of two different forms:

- A DataInstance
- A DI List

A **DataInstance** is the result of a query consisting of objects. A **DataInstance** is the capture of only one result, although the result may consist of many items requested by the **MetaObject**.

A **DIList** is the capture of more than one **DataInstance**.

The following diagram shows the difference between a **DataInstance** and a **DIList**.

Project Name					
MetaObject Name					
Item 11	Item 21	Item 31	Item 41	Item 51	Item 61
Item 21	Item 22	Item 32	Item 42	Item 52	Item 62
Item 31	Item 23	Item 33	Item 43	Item 53	Item 63
Item 41	Item 24	Item 34	Item 44	Item 54	Item 64
Item 51	Item 25	Item 35	Item 45	Item 55	Item 65

### **DataInstance**

A **DataInstance** is represented by the left hand column. The **DataInstance** consists of a **Project Name** and a **MetaObject** name plus results of the query as defined by the **MetaObject**. In this case, these are items 11, 21, 31, 41, 51, which simply bear a relationship to the **MetaObject**, but may have come from diverse and unrelated tables and sources.

### **DI List**

A **DI List** is represented by all of the table including the **Project Name** and **MetaObject** Name, and including all the items shown, items which embody six **DataInstances**. The columns, as represented here, can be addressed separately, as numbered columns or as "Temp" items, so that results from a **DIList** can be re-formed to **DataInstance** data. ("Temp" items are an arbitrarily defined parameter in a **DIList** method.)

The DataInstance interface returns a result depending on the type of connection - 1 to 1 or 1 to Many. 1 to 1 returns a Data Instance. 1 to Many returns a DI List.

Some data instance implementations are immutable (they cannot be changed within the current programming framework) and will throw an Unsupported Operation Exception if a change is made to them.

## **Methods**

getValue

(EQ 62)

public Object getValue(String property)throws RemoteException

### *Description*

Gets the value of a property in the data base.

### *Parameters*

property - an item in the data base

### *Return Value*

Returns the value of the property within the object.

### *Exceptions*

RemoteException

getAllValues

(EQ 63)

public Map getAllValues()throws RemoteException

*Description*

Gets all the values of properties in the data base.

*Return Value*

Returns all values within the map.

*Exceptions*

RemoteException

getProject

(EQ 64)

public String getProject()

*Description*

Gets the Project file.

*Return Value*

Returns the project file



getMeta

(EQ 65)

public String getMeta()

*Description*

Gets the MetaObject file.

*Return Value*

Returns the MetaObject

getChildren

(EQ 66)

public Object getChildren(String connection)throws  
RemoteException

*Description*

Gets the children of the specified MetaObject file.

*Parameters*

connection - nativeIdentifier

*Return Value*

Returns the children metaObject files.

*Exceptions*

RemoteException

(EQ 67)

(EQ 68)

(EQ 69)

(EQ 70)

getAllConnections

(EQ 71)

public Map getAllConnections()throws RemoteException

*Description*

Gets the introspection of all the data source connections.

*Return*

Returns a list of all the data source connections.

*Exceptions*

RemoteException

isRemote

(EQ 72)

public boolean isRemote()

*Description*

Determines if a process is remote.

*Return*

Connects to a remote process.

---

***DIList***

---

package meta.dataInstance

**Description**

The DIList interface contains a list of data instance classes and a data structure that stores them in an efficient way. Get() and iterators return DataInstance implementations. Implementations of this class are serialized.

For efficiency in iterating, a single temporary Object that can be returned every time with different values. Changes to this object should be reflected in the corresponding values in the list.

**Methods**

getTemp

(EQ 73)

public DataInstance getTemp(int i)

***Description***

The Temp method returns data in the same memory space without the need to use memory for the whole DI List structure.

***Parameters***

temp – an integer that specifies a field number (column) in the DIList

***Return***

Returns the contents of the field (column) specified.

---

## *DataSource*

---

package data

The DataSource interface defines and manages the connection to a data source. AltoStudio configures and accesses introspection data from the underlying data sources during the DeveloperSessions. At Runtime, methods create and manage a connection to the data source.

AltoStudio sets the connection properties of the data source via AltoStudio. It also obtains the following introspection information about DataSet(s), Field(s), Connections From DataSet(s) and Connections To DataSet(s). It adds and removes DataSets. It also gets new configurables and configurable GUI models.

At Runtime, it gets and returns the DataSets selected. It connects to the data source and implements default connection properties. It runs a query based on queryData (only at runtime) and also runs a dynamic query.

### **Methods**

setConnectionProp

(EQ 74)

public void setConnectionProp(String name, Object value)

#### *Description*

Allows the user to set the connection properties.

#### *Parameters*

name

value

#### *Return Value*

Returns the connection

getDataSetsIdentifiers

(EQ 75)

public Iterator getDataSetsIdentifiers()

*Description*

Gets the DataSource content. These DataSets are in the object. For JDBC, the DataSetIdentifier is the table name (String).

*Return Value*

Returns the cached DataSets in the DataSource

getDataSet

(EQ 76)

public Iterator getDataSets()

*Description*

Gets the DataSets.

*Return Value*

Returns the DataSet

getDataSet

(EQ 77)

public DataSet getDataSet(Object dataSetIdentifier)

*Description*

Gets the DataSet identified by DataSetIdentifier.

*Parameters*

dataSetIdentifier – the name of the DataSet

*Return Value*

Returns the dataSet. Returns null if it does not exist

DataSet

(EQ 78)

```
public void addDataSet(DataSet dataSet)//, String name)
```

*Description*

Adds a Dataset.

*Parameters*

dataSet - the name of a DataSet (String)

*Return Value*

Adds a DataSet. The Name can be null. In this case, the system finds a name

removeDataSetByName

(EQ 79)

```
public void removeDataSetByName(String name)
```

*Description*

Removes a DataSet from list of connections.

*Parameters*

name - name of DataSet (string)

*Return Value*

Removes DataSet

**removeDataSet****(EQ 80)****public void removeDataSet(Object nativeIdentifier)*****Description***

Removes a DataSet from the list of connections.

***Parameters***

nativeIdentifier - identifying label

***Return Value***

Removes DataSet

**getAllTypes****(EQ 81)****public String[] getAllTypes() throws ConnectionException*****Description***

This feature provides for introspection. It returns all the DataSets requested from the native source. They are not cached. The Studio never calls these DataSets directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.

***Return Value***

Returns the DataSet names. Returns null if there are no DataSets.

***Exceptions***

ConnectionException

allDataSetIdentifiers

(EQ 82)

public List allDataSetIdentifiers() throws BadQueryException,  
ConnectionException

*Description*

This feature provides for introspection. It returns all the DataSets requested from the native source. They are not cached. The Studio never calls these DataSets directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.

*Return Value*

Returns objects that are the native identifiers of the DataSets for a database. A native identifier will be a String with the table name.

*Exceptions*

BadQueryException, ConnectionException

allDataSetIdentifiers

(EQ 83)

public List allDataSetIdentifiers(String type) throws BadQueryException,  
ConnectionException

*Description*

This feature provides for introspection. It returns all the DataSets requested from the native source. They are not cached. The Studio never calls these DataSets directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.



*Parameters*

type - type of DataSetIdentifiers

*Return Value*

Returns all DataSetIdentifiers.

*Exceptions*

BadQueryException, ConnectionException

dataSet

(EQ 84)

public DataSet dataSet(Object nativeIdentifier) throws BadQueryException, ConnectionException

*Description*

*Parameters*

nativeIdentifier -the name of the DataSet

*Return Value*

The studio have to connect the dataset to its own DataSource

*Exceptions*

BadQueryException, ConnectionException

allFieldIdentifiers

(EQ 85)

public List allFieldIdentifiers(Object dataSetIdentifier) throws  
BadQueryException, ConnectionException

*Description*

This feature provides for introspection. It returns all the Fields requested from the native source. They are not cached. The Studio never calls these Fields directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.

*Parameters*

dataSetIdentifiers - the identifier of the DataSet for which the fields are requested

*Return Value*

Returns all FieldIdentifiers.

*Exceptions*

BadQueryException, ConnectionException

field

(EQ 86)

public Field field(Object dataSetIdentifier, Object fieldIdentifier)  
throws BadQueryException, ConnectionException

*Description*

Requests a Field.

*Parameters*

**dataSetIdentifier** - the identifier of the DataSet for which the fields are requested

**fieldIdentifier** - the identifier of the field

*Return Value*

Returns a field from a specified DataSet.

*Exceptions*

BadQueryException, ConnectionException

**allDataSets**

**(EQ 87)**

**public List allDataSets()** throws BadQueryException, ConnectionException

*Description*

This feature provides for introspection. It returns all the DataSets requested from the native source. They are not cached. The Studio never calls these DataSets directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.

*Return Value*

Returns a list of all DataSets.

*Exceptions*

BadQueryException, ConnectionException

allFields

(EQ 88)

public List allFields(Object dataSetIdentifier) throws BadQueryException, ConnectionException

*Description*

This feature provides for introspection. It returns all the DataSets requested from the native source. They are not cached. The Studio never calls these DataSets directly. It calls the DeveloperSession, that calls them. Only the server accesses clients. The DataSource is a factory for Field and DataSet classes.

*Parameters*

dataSetIdentifiers – the identifier of the DataSet for which the fields are requested

*Return Value*

Returns a list of all the Fields.

*Exceptions*

BadQueryException, ConnectionException

connectionsFrom

(EQ 89)

public List connectionsFrom(Object dataSetNativeIdentifier)  
throws BadQueryException, ConnectionException

*Description*

Obtains a list of connections from a specified DataSet.

*Parameters*

dataSetNativeIdentifier – DataSet identifier

*Return Value*

Returns a list connections from the DataSet.

*Exceptions*

BadQueryException, ConnectionException

connectionsTo

(EQ 90)

public List connectionsTo(Object dataSetNativeIdentifier) throws  
BadQueryException, ConnectionException

*Description*

Obtains a list of connections to a specified DataSet.

*Parameters*

dataSetNativeIdentifier – DataSet identifier

*Return Value*

Returns a list connections from the DataSet (ConnectionInfo Objects).

### *Exceptions*

BadQueryException, ConnectionException

runQuery

(EQ 91)

public Object runQuery(Object queryData) throws BadQueryException, ConnectionException

### *Description*

This function is always called at runtime. For efficiency, it always returns the same one. This class is guaranteed not to have more than one thread executing it, same with the DynamicQuery. This is why it is thread safe to reuse the same object

### *Parameters*

queryData – the values of data in the query object

### *Return Value*

Returns an unknown query that is specific to the DataSource implementation.

### *Exceptions*

BadQueryException, ConnectionException

getDynamicQuery

(EQ 92)

public DynamicQuery getDynamicQuery()

*Description*

Gets a Dynamic Query.

*Return Value*

Returns DynamicQuery

getNewConfigurable

(EQ 93)

public ConfigurableQuery getNewConfigurable()

*Description*

Gets a new Configurable Query

*Return Value*

Returns a new Configurable Query

getConfGuiModel

(EQ 94)

public ConfigurableGuiModel getConfGuiModel(String confName)

*Description*

Gets a Configurable GUI Model.

*Parameters*

confName – model name

*Return Value*

Returns Configurable GUI Model

getAllConfigurables

(EQ 95)

public List getAllConfigurables()

*Description*

Gets all Configurables

*Return Value*

A list of all the ConfigurableGuiModel names (Strings)

connectAndConfigure

(EQ 96)

public void connectAndConfigure() throws ConnectionException

*Description*

Connects and configures data sources.

*Return Value*

Connection at runtime. Executed once per class when it is loaded. If more than one class is loaded with the same connection parameters, the implementation handles connections.

*Exceptions*

ConnectionException



**Connect****(EQ 97)**

```
public void connect() throws ConnectionException
```

*Description*

For creating a connection by AltoStudio. In case the some extra configuration information is required in the data source for later use (like runtime), this function should do that.

*Return Value*

Connects to data source.

*Exceptions*

ConnectionException

**Disconnect****(EQ 98)**

```
public void disconnect() throws ConnectionException
```

```
getDefaultConnectionProps
```

```
public Map getDefaultConnectionProps()
```

*Description*

Gets the default connection properties.

*Return Value*

Returns connection prop with some default props when it makes sense. Example: JDBC driver makes sense to have a default, password not accepted.

### *Exceptions*

#### ConnectionException

getConfig

(EQ 99)

public DataSourceConfigModel getConfig()

#### Description

Gets the configurable model.

#### Return Value

Returns the configurable model

---

## ***DataSet***

package data

### **Description**

The DataSet interface selects the DataSource from which a DataSet is constructed. It calls and names DataSets and adds, removes and maps all the fields within a DataSet.

A DataSet is a table or sequence which embodies a relationship between two or more variables, which maps to one or more data sources. A DataSet consists of Fields.

A Field is a single element of a DataSet such as a column or row in a table. DataSets are commonly found in RDBMSs, but they may be used to represent any underlying data source, whether web, file or multimedia sources. DataSets may be used, in principle, to map to any actual data source. One DataSet can be used to create several MetaObjects.

## Methods

getSource

(EQ 100)

public DataSource getSource()

### *Description*

Gets the DataSource to which the query is addressed.

### *Return Value*

Returns the selected data source. This is not user defined.

setSource

(EQ 101)

public void setSource(DataSource ds)

### *Description*

Selects the DataSource to which the query is addressed.

### *Parameters*

data source – nativeIdentifier

### *Return Value*

Returns the selected data source. The nativeIdentifier identifies the DataSource. JDBC specifies the table name (String) as the nativeIdentifier. This is user defined.

getNativeIdentifier

(EQ 102)

public Object getNativeIdentifier()

*Description*

Finds the nativeIdentifier of the DataSource.

*Return Value*

Returns the nativeIdentifier of the DataSource.

getType

(EQ 103)

public String getType()

*Description*

Gets the data type of the DataSet content.

*Return Value*

Returns the data type of the DataSet content. In JDBC, the dataSetIdentifier is the Column name (String). This returns the cached Fields in the DataSet.

getFieldsIdentifiers

(EQ 104)

public Iterator getFieldsIdentifiers()

*Description*

Gets the Identifiers of the Fields in the DataSet.

*Return Value*

Returns all the Fields in the DataSet.

getFields

(EQ 105)

public Iterator getFields()

*Description*

Gets the Fields in the DataSet.

*Return Value*

Returns the Fields in the DataSet.

getField

(EQ 106)

public Field getField(Object nativeIdentifier)

*Description*

Gets a Field identified by its nativeIdentifier. In JDBC, this would be a column name.

*Parameters*

nativeIdentifier – nativeIdentifier of the field

*Return Value*

Returns a field. The name can be null, then the system finds a name

addField

(EQ 107)

```
public void addField(Field field)//Object nativeIdentifier, String  
name)
```

*Description*

Adds a field to a DataSet.

*Parameters*

nativeIdentifier – nativeIdentifier

name – name of the field (string)

*Return Value*

Returns a DataSet with a field added.

removeFieldByName

(EQ 108)

```
public void removeFieldByName(String name)
```

*Description*

Removes a field from a DataSet.

*Parameters*

name – name of field

*Return Value*

Returns a DataSet with a Field removed.

**removeField**

**(EQ 109)**

**public void removeField(Object nativeIdentifier)**

*Description*

Remove a field from the DataSet

*Parameters*

nativeIdentifier – native identifier of the field to be removed

*Return Value*

Returns a DataSet with one field removed.

**removeAll**

**(EQ 110)**

**public void removeAll()**

*Description*

Removes all the fields of the DataSet.

*Return Value*

Returns a DataSet with no fields.

---

## *Field*

package data

### **Description**

The Field interface is used to specify queries which relate to Fields, a single element in a DataSet. It allows the user to define a name for a Field, and it gets the Field. It gets the native identifier for the Field, its DataSet and its DataSource connection. The data types for items in the Field are defined here. They may be Java Types or SQL Types (of the java.sql class). A Field is tagged as PK if it is the Primary Key of a DataSet. The default value of a Field is null.

In implementations of this interface, it's necessary to make sure that the data types of the items in the Field are uniquely defined. This is the place to address any Field issues as they might relate to other classes and packages.

### **Methods**

setName

(EQ 111)

public void setName(String name)

#### *Description*

Specifies user defined name.

#### *Parameter*

name - the name of a Field

#### *Return Value*

Returns Field name.



getName

(EQ 112)

public String getName()

*Description*

Gets a specified field.

*Return Value*

Returns the Field.

getNativeIdentifier

(EQ 113)

public Object getNativeIdentifier()

*Description*

Connects to a DataSource.

*Return Value*

Returns the NativeIdentifier for the data source. For JDBC, it is the table name (string).

getDataSet

(EQ 114)

public DataSet getDataSet()

*Description*

Connects to a DataSet.

*Return Value*

Returns the DataSet name.

setDataSet

(EQ 115)

public void setDataSet(DataSet ds)

*Description*

Sets the data connector interface to a DataSet.

*Return Value*

Returns the DataSet native identifier.

javaType

(EQ 116)

public int javaType()

*Description*

Initializes parameters to Java types as shown below.

*Java types specified as ints*

JAVA\_UNDEFINED = 0,

JAVA\_STRING = 1, (int is used for character, byte and integer)

JAVA\_INT = 2,

JAVA\_FLOAT = 3,

JAVA\_DOUBLE = 4,

JAVA\_LONG = 5,

JAVA\_BINARY = 6,

JAVA\_BOOLEAN = 7,

JAVA\_DATE = 8, (These data sources require processing at the Field implementation level)

---

**Field**

---

JAVA\_OBJECT = 9,

JAVA\_STREAM = 10,

JAVA\_IMAGE = 11

*Return Value*

Returns selected parameter(s) as Java types.

sqlType

(EQ 117)

public int sqlType()

*Description*

Initializes parameters to SQL types as defined in java.sql.Types class.

*Return Value*

Returns selected parameter(s) as SQL types, as specified.

isPrimaryKey

(EQ 118)

public boolean isPrimaryKey()

*Description*

Specifies the PrimaryKey of the Field.

*Return Value*

Returns the Primary Key.

defaultValue

(EQ 119)

public Object defaultValue()

*Description*

Sets a default value to the Field.

*Return Value*

Returns null.

---

## **BasicQuery**

package data

### **Description**

The BasicQuery interface is used to set data types for a query during runtime. It uses the JDBC API set of data types. The Basic Query interface is transparent and is used by both DynamicQuery and ConfigurableQuery. It underlies the function of these two types of query.

During execution, this class functions as a result set. QueryAction-Logic might use this interface where calculations to be performed on the results of a query require a specific data type.

### **Methods**

getSource

(EQ 120)

public DataSource getSource()

*Description*

Gets the data source selected to apply the data type selection.

*Return Value*

Returns the data instance or DIList with the data types set.

setSource

(EQ 121)

public void setSource(DataSource source)

*Description*

Sets the data source selected for the data type selected.

*Parameters*

source – a data source

*Return Value*

Returns the data instance or DIList with the data type set.

**Data Types**

getBinaryStream — public InputStream getBinaryStream(String name) throws ConnectionException

getBoolean — public boolean getBoolean(String name) throws ConnectionException

getByte — public byte getByte(String name) throws ConnectionException

getBytes — public byte[] getBytes(String name) throws ConnectionException

getDate — public Date getDate(String name) throws ConnectionException

getDouble — public double getDouble(String name)throws ConnectionException

getFloat — public float getFloat(String name)throws ConnectionException

getInt — public int getInt(String name)throws ConnectionException

getLong — public long getLong(String name)throws ConnectionException

getObject — public Object getObject(String name)throws ConnectionException

getShort — public short getShort(String name)throws ConnectionException

getString — public String getString(String name)throws ConnectionException

---

## *DynamicQuery*

package data

### **Description**

A DynamicQuery is a request for information which can access a range of possible data. It accepts parameters, names, parents, etc. to limit the query search. It is a stateless program. Information is not kept.

A DynamicQuery performs anonymous query functions. That is to say, query functions that have not been specified by AltoStudio.

There are two types of method in the DynamicQuery interface.

- `getSingle` returns a `DataInstance`.
- `getMany` returns a `DIList`

The method returns a result associated with a `metaObject`. The rest of the method implementation is related to the conditionally that defines the data requested. The number of expected results is determined here.

The Map contains conditions. These map to the `metaObject` or `Field` in the data base. Each key/value pair in it represents the `metaObject` property and the user's value for the query, in that order. Properties are the items in the `metaObject` or `Field` requested. Properties may be any data type, as long as it is allowed by the `BasicQuery` interface. `QueryActionLogic` which is specified may be written in Java or SQL.

A `DynamicQuery` may be defined in relation to parent and child as defined by the object connection properties, so that the data selected may be business logic from `metaObjects` and `Fields` from past, present and future selections with additional conditionally provided.

String properties are included in the `DynamicQuery` when the result (`DataInstance` or `DIList`) should not include those properties. Those properties are excised from the `metaObject`. If all properties in the `metaObject` are to be included, it sends null.

Two types of exception typically occur with the `DynamicQuery`.

- A `Connection Exception` indicates a problem in the underlying `DataSource`.
- A `BadQueryException` indicates a problem with the parameters sent to the function (like wrong or inappropriate names).

The `DynamicQuery` interface provides the place to initiate custom queries to user specification.

## Methods

getSingle

(EQ 122)

public DataInstance getSingle(MetaObject meta, Map conditions, String[] selectProperties, QueryLogicAction logic) throws ConnectionException, BadQueryException

### *Description*

Performs a DynamicQuery to meet specified conditions returning a DataInstance as a result.

### *Parameters*

meta – the name of a metaObject (string)

conditions - numeric data types which specify a map of the part of the table in which to look

selectProperties – item(s) in the data base

logic – a logical condition such as mathematical or Boolean relationship

### *Return Value*

Returns a DataInstance.

### *Exceptions*

ConnectionException, BadQueryException



getMany

(EQ 123)

```
public DIList getMany(MetaObject meta, Map conditions, String[]
selectProperties, QueryLogicAction logic)throws ConnectionExcep-
tion, BadQueryException
```

*Description*

Performs a DynamicQuery to meet specified conditions returning a DIList as a result.

*Parameters*

meta - the name of a metaObject (string)

conditions - numeric data types which specify a map of the part of the table in which to look

selectProperties - item(s) in a data base

logic - a logical condition such as mathematical or Boolean relationship

*Return Value*

Returns a DIList (more than one DataInstance).

*Exceptions*

ConnectionException, BadQueryException

getSingleByParent

(EQ 124)

```
public DataInstance getSingleByParent(Object parent, String  
parentProp, MetaObject childMeta, String childProp, String[]  
selectProperties, QueryLogicAction logic)throws ConnectionExcep-  
tion, BadQueryException
```

*Description*

Performs a DynamicQuery to meet specified conditions returning a DataInstance as a result.

*Parameters*

parent - the name of the parent metaObject

parentProp - an item in the parent metaObject table

childMeta - the name of a child metaObject

childProp - the name of an item of the data base in the child metaObject

selectProperties - item(s) in the data base

logic - a logical condition such as mathematical or Boolean relationship

*Return Value*

Returns a DataInstance.

*Exceptions*

ConnectionException, BadQueryException

getManyByParent

(EQ 125)

```
public DList getManyByParent(Object parent, String parentProp,  
MetaObject childMeta, String childField, String[] selectProperties,  
QueryLogicAction logic)throws ConnectionException, BadQueryEx-  
ception
```

*Description*

Performs a DynamicQuery to meet specified conditions returning a DList as a result.

*Parameters*

parent – the name of the parent metaObject

parentProp – an item in the parent metaObject table

childMeta – the name of a child metaObject

childField – the name of a Field in the child metaObject

selectProperties – item(s) in the data base

logic - a logical condition such as mathematical or Boolean relationship

*Return Value*

Returns DI List.

*Exceptions*

ConnectionException, BadQueryException

---

## *ConfigurableQuery*

---

package data

### **Description**

The ConfigurableQuery interface defines queries specified by AltoStudio using the connection diagram.

It supports all types of DataSource including SQL. Each ConfigurableQuery configures a single QuerySession instance.

The ConfigurableQuery interface executes the following features.

- It allows the user to name and call metaObjects and selects relevant parameters for query.
- It executes the join features between metaObjects and Fields, as specified.
- It executes operations on metaObjects such as ordering of items and application of conditionally.
- It executes custom queries defined in SQL.
- It returns DataInstances and DILists as determined, if appropriate, by conditionally.

Most query definition by AltoStudio is facilitated by the visual development environment, but a programming window is also provided. All these features defined by AltoStudio are executed by the ConfigurableQuery.

A parameter is required when an operation is defined that needs external data. The parameter selection restricts the query to parameter names like age, name, year, etc. In this case, the word "parameter" refers to any set of properties whose values determine the characteristics or behavior of something. It is an arbitrary constant whose value characterizes a member of a system.

Example sql: select...where id=5 needs 5 as a parameter, so setEquals condition needs a parameter name.

Master metaObjects are distinguished from those which will be joined to them by using the convention that the prefix "my" is inserted before the metaObject in the case of a master, and "other" inserted in front of the metaObject in the case of the metaObject(s) which will be joined to them. Similarly, the convention is applied to Fields and Properties. Thus, we have myMeta and myField, and otherMeta and otherField.

The ProgrammableQuery class implements ConfigurableQuery. The ProgrammableQuery function is entered from the Programming Window in AltoStudio. It essentially implements all the ConfigurableQuery properties.

## **Methods**

setMetaObject (EQ 126)

public void setMetaObject(MetaObject meta)

### *Description*

Specifies a metaObject to run a query. This is a user defined metaObject with user defined name.

### *Parameters*

meta – name of the metaObject (string)

### *Return Value*

Returns a DataInstance of a specified metaObject.

selectedParameters (EQ 127)

public void selectedParameters(String[] parameterNames)

### *Description*

Restricts the query to parameter names like age, name, year, etc. In this case, the word "parameter" refers to any set of properties whose values deter-

mine the characteristics or behavior of something. It is an arbitrary constant whose value characterizes a member of a system.

*Parameters*

parameterNames – names of parameters not to be returned (strings)

*Return Value*

Returns all parameters in the result except those specified.

setSingleJoinObject

(EQ 128)

```
public void setSingleJoinObject(String parameter, MetaObject otherMeta, String otherPropName)throws BadQueryException,
UnsupportedQueryException
```

*Description*

Joins to other metaObject from the same DataSource

*Parameters*

parameter – the name of the parameter

otherMeta – the name of the second metaObject, the one to be joined to the master

otherPropName – the property name of the second metaObject

*Return Value*

Returns a DataInstance or DIList of the metaObjects specified in the join.

setMultiJoinObject

(EQ 129)

```
public void setMultiJoinObject (String parameter, String prop-
Name, MetaObject otherMeta,
String otherPropName)throws BadQueryException, Unsupport-
edQueryException
```

*Description*

Joins to other metaObjects from multiple DataSources.

*Parameters*

parameter – the name of the parameter

propName – the property name of the master metaOb-  
ject in the join

otherMeta – the name of the second metaObject, the one  
to be joined to the master

otherPropName – the property name of the second  
metaObject

*Return Value*

Returns a DataInstance or DIList of the metaObjects  
specified in the join.

setSingleJoinField

(EQ 130)

public void setSingleJoinField(String parameter, Field myFiled, Field otherField)throws BadQueryException, UnsupportedQueryException

*Description*

Join to other DataSet from the same DataSource myField is in my DataSource (my = the MetaObject) otherField is the other objects join column for now we can not implement this.

*Parameters*

parameter – the name of the parameter

myField – the name of the Field in the master metaObject

otherField – the name of the Field to be joined to the master

*Return Value*

Returns a DataInstance or DIList of the Fields specified in the join.

setMultiJoinField

(EQ 131)

public void setMultiJoinField(String parameter, Field myField, Field otherField)throws BadQueryException, UnsupportedQueryException

*Description*

Joins multiple fields together.



*Parameters*

parameter - the name of the parameter

myField - the name of the Field in the master metaObject

otherField - the name of the Field to be joined to the master

*Return*

Returns a table with a Field from the master metaObject and a field from the metaObject to be joined, which excludes items with a value of a specified parameter.

addOrder

(EQ 132)

public void addOrder(String propName) throws BadQueryException, UnsupportedOperationException

*Description*

Adds order by property name to the resulting table.

*Parameters*

propName - the property name of the metaObject or Field.

*Return*

Returns an ordered table in the result.

addOrder

(EQ 133)

public void addOrder(Field orderField)throws SQLException,  
UnsupportedQueryException

*Description*

Adds order by property name to the resulting Field.

*Parameters*

orderField - SQL order command

*Return*

Returns an ordered Field (column) in the result.

addEqualsCondition

(EQ 134)

public void addEqualsCondition(String parameter, String prop-  
Name)throws SQLException, UnsupportedQueryException

*Description*

Applies an equality condition to properties among a  
range of the selected parameter.

*Parameters*

parameter - the name of the parameter

propName - the property name of the metaObject or  
Field.

*Return Value*

Returns results which applies the equals condition.

addEqualsCondition

(EQ 135)

```
public void addEqualsCondition(String parameter, MetaObject otherMeta, String myPropName, String otherPropName)throws BadQueryException, UnsupportedQueryException
```

*Description*

Applies the equals condition between two properties in two different metaObjects. This function gets a parameter a dataInstance that is described by otherMeta metaObject. It takes a field from this object and matches it to a field in the query.

*Parameters*

parameter - the name of the parameter

otherMeta - the name of the metaObject to be joined to the master

myPropName - the property name of the metaObject or Field.

otherPropName - the property name of the metaObject or Field.

*Return Value*

Returns results with the equals condition applied to both metaObjects.

addInCondition

(EQ 136)

public void addInCondition(String parameter, String propName)throws BadQueryException, UnsupportedOperationException

*Description*

Tests to determine if a property lies within a parameter range. The feature looks to find, identify and return one or more items within a given range.

*Parameters*

parameter - - the name of the parameter

propName - the property name of the metaObject or Field.

*Return Value*

Returns results within the parameter range that contain the property.

addInCondition

(EQ 137)

public void addInCondition(String parameter, MetaObject otherMeta, String myPropName,

String otherPropName)throws BadQueryException, UnsupportedOperationException

*Description*

Tests to determine if a property lies within a parameter range in two metaObjects. The feature looks to find, identify and return one or more items within a given range.

*Parameters*

parameter - the name of the parameter

otherMeta - the name of the metaObject to be joined to the master

myPropName - the property name of the master metaObject or Field.

otherPropName - the property name of the metaObject or Field to be joined.

*Return Value*

Returns a result within the parameter range that contain the property in two metaObjects.

addRangeCondition

(EQ 138)

public void addRangeCondition(String upperBoundParameter,  
String lowerBoundParameter,

String propName)throws BadQueryException, UnsupportedQueryException

*Description*

Applies a range condition to the property. Determines if the property lie within the upper and lower bound values.

*Parameters*

upperBoundParameter - a numeric data type (e.g. int, double, short, float, etc.)

lowerBoundParameter - a numeric data type (e.g. int, double, short, float, etc.)

propName - the property name of the metaObject or Field.

*Return Value*

Returns a result (DataInstance or DIList) with the range condition applied.

addRangeCondition

(EQ 139)

public void addRangeCondition(String upperBoundParameter,  
String lowerBoundParameter,

Field myField, MetaObject upperBoundMetaObject, Field upper-  
BoundField, MetaObject lowerBoundMetaObject, Field lower-  
BoundField)throws BadQueryException,  
UnsupportedQueryException

*Description*

Applies a range condition to metaObjects and Fields which have themselves had upper and lower limits applied to their range of values. To summarize, the values of either a metaObject and/or Field are bound within two values specified by upper and lower bound values for each object. An additional range condition is applied to the result.

*Parameters*

upperBoundParameter - a numeric data type (e.g. int, double, short, float, etc.)

lowerBoundParameter - a numeric data type (e.g. int, double, short, float, etc.)

myField - the master metaObject

upperBoundMetaObject – a numeric data type (e.g. int, double, short, float, etc.) for the metaObject

upperBoundField – a numeric data type (e.g. int, double, short, float, etc.) for the Field

lowerBoundMetaObject – a numeric data type (e.g. int, double, short, float, etc.) for the metaObject

lowerBoundField – a numeric data type (e.g. int, double, short, float, etc.) – for the Field

*Return Value*

Returns a result (DataInstance or DIList) with the range condition applied.

addRegExpCondition

(EQ 140)

public void addRegExpCondition(String parameter, String propName) throws UnsupportedOperationException

*Description*

Applies a Regular Expression condition. This condition verifies the format of the items, not the value.

*Parameters*

parameter - the name of the parameter

propName - the property name of the metaObject or Field.

*Return Value*

Returns a result with a common verified format.

setCustomQuery

(EQ 141)

public void setCustomQuery(Object customQuery)throws BadQueryException, UnsupportedOperationException

*Description*

Custom query can be a SQL String with ? for parameters. In this case, the parameters are set with a string which represents their serial number, starting with "1" (and then "2", "3",....).

*Parameters*

customQuery – a logical relationship written in SQL

*Return Value*

Returns the result of a custom query.

getParentParameters

(EQ 142)

public String[] getParentParameters()

*Description*

This feature provides information to AltoStudio to build the state object in layout. This is not a runtime function.

*Return Value*

Returns the parent parameters that are required for the execution of the query.



getSingle

(EQ 143)

public DataInstance getSingle(Map conditions, String[] selectProperties, DataInstance parent, QueryLogicAction logic) throws ConnectionException, BadQueryException

*Description*

Gets a DataInstance within a specified map of the metaObject, with common property, parent and logical formula applied.

*Parameters*

conditions - numeric data types which specify a map of the part of the table in which to look

selectProperties - an item in the data base (any data type, usually a string)

parent - the name of the parent metaObject

logic - a logical operation (mathematical or Boolean)

*Return Value*

Returns a DataInstance based on the conditionally applied.

getMany

(EQ 144)

public DIList getMany(Map conditions, String[] selectProperties, DataInstance parent, QueryLogicAction logic) throws ConnectionException, BadQueryException

*Description*

Gets a DIList within a specified map of the metaObject, with common property, parent.

*Parameters*

conditions – numeric data types which specify a map of the part of the table in which to look

selectProperties – an item in the data base (any data type, usually a string)

parent – the name of the parent metaObject

*Return Value*

Returns a DIList based on the conditionally applied.

---

## *QueryLogicAction*

package meta

### **Description**

The QueryActionLogic class executes logical operations on the results of the query. It is executed by the session beans and is associated with MetaObjects on the entity beans.

It applies a logic model to DataInstances and DILists. It takes results (data) returned from the data base and performs logical operations such as calculations or booleans to provide an end-result that can be subsequently sent to the client. It can also re-organize the data to meet the client presentation requirements.

QueryLogicAction selects methods or objects (selected by next or get) and puts them in a chain for execution by the program. AltoStudio defines methods or objects to be used in the chain. Either, the methods or objects can originate from AltoStudio itself, from classes like java.math or from elsewhere, as appropriate.

QueryActionLogic provides in-stream processing of data removed from operations in the data base management system.

## **Methods**

next

(EQ 145)

public QueryLogicAction next

### *Description*

Identifies the name of a method or object to be used by QueryLogicAction.

### *Parameters*

next - the name of the next method or object (a string)

### *Return Value*

Returns the next QueryLogicAction object.

getName

(EQ 146)

public abstract String getName()

*Description*

Gets the name of a method or object from AltoStudio.

*Parameter*

name – a name of a method or object (a string)

*Return Value*

Returns a name for the studio

chain

(EQ 147)

public void chain(QueryLogicAction next)

*Description*

Chains logic classes one after another.

*Parameter*

next – name of the next method or object (a string)

*Return Value*

Returns the result of the logical operation

getNext

(EQ 148)

public QueryLogicAction getNext()

*Description*

Gets the next method or object.

*Return Value*

Returns the result of a logical operation.

beforeAll

(EQ 149)

public void beforeAll(Object parent, MetaObject myMeta, Basic-  
Query queryResults)

*Description*

Initializes a logical operations prior to execution.

*Parameters*

parent - a Data Instance of the parent, or a List of  
DataInstances (a string)

myMeta - the name of a metaObject, which identifies  
the parent and the QueryLogicAction methods or  
objects to be performed

queryResults - the name of the end result

*Return Value*

Returns a value for the result of each step in a  
sequence.

afterAll

(EQ 150)

public void afterAll(List allData)

*Description*

Performs a logical operation on a set of data.

*Parameters*

allData - all data specified in the list

*Return Value*

Returns the result of one sequential logical operation of the same data instance.

afterIteration

(EQ 151)

public void afterIteration(LocalDataInstance data)

*Description*

Performs loop operations of beforeAll and afterAll to produce an end-result.

*Parameters*

data - the data in the same process to be processed

*Return Value*

Returns the result of a loop.

getConfiguration

(EQ 152)

public abstract QueryLogicActionModel getConfiguration()

*Description*

Only called from the studio

*Returns*

Returns new myConfigurationInnerClass

---

*Exceptions*

The following exceptions are defined as part of the AltoServer implementation.

**ConfigurableQueryException**

**ConnectionException** is thrown when the class cannot connect to the data base.

**BadQueryException** is thrown when the user asks for something that isn't possible, such as the content of a table that doesn't exist.

**RemoteException** is thrown when the data requested is from a process other than the one that references the data.

**CreateException** is thrown when the object which is being created does not meet logical criteria for creation.

**UnsupportedQueryException** is thrown when trying to set the query function on an instance of QuerySession if the query is not supported.

**QuerySequenceException** is thrown when QuerySession can support some single queries but not a combination of queries.

**ProjectException** or **ProjectAccessException** is thrown when the project can't be found or if the user does not have access rights to it.

**StateException** is thrown when querying on states that do not exist, not sending the appropriate parameters that the function expects, etc.



The Glossary contains a list of terms commonly used in the development of AltoWeb applications.

TABLE 4. Glossary

Name	Definition
AltoClient	A Java executable that runs as a standalone application or as an applet. It can interface to AltoServer and present information structures in multiple views by mapping them on three dimensional shapes or HTML.
AltoCube	A client based implementation of AltoClient which maps to a cube.
AltoServer	A program which manages the client/data source functionality.
AltoServlet	A server program that connects to a client or data source application.
AltoSphere	A client based implementation of AltoClient which maps to a sphere.
AltoStudio	A visual development environment that provides a connection to AltoServer, browses different data sources, builds data maps, and programs visual and interactive features of the client.

TABLE 4. Glossary (Continued)

Name	Definition
Applet	A small program, usually a part of a larger application such as a web page, which can carry out specific tasks such as interactive animation or immediate calculations.
Application Server	A server program in a computer in a distributed network that provides the business logic for an application program.
Authentication	The process of determining whether someone or something is who or what it is declared to be.
Bands	Describes the logical relationships between items in the context window represented by various visual properties.
Basic Query	An interface class which sets the data types used by a connection for the purpose of processing queries. It uses standard Java types. Dynamic and Configurable Queries use its set up characteristics and are sub-types.
Cardinality	Specifies the characteristics of connections between MetaObjects. They may have a cardinality of 1 to 1 or 1 to N. 1 to 1 cardinality defines a connection from one MetaObject to only one other MetaObject and returns a DataInstance as a result. 1 to N cardinality defines more than one connection between MetaObjects and returns a DI List as a result.
Clients	Users of the server that have an identity.
Connections	A link to a specific data sources using specified electrical and protocol characteristics. Queries and their results use connections.
Configurable Query	An interface for passing data processing requests created by AltoStudio. It has predetermined requests defined in AltoStudio by the connection diagram.
Container	A series of programs that are compatible and run together based in a common framework.
Content	The content is the end-result of the AltoWeb experience. Control windows provide detailed information about an entity. It may contain elements such as text, static images, streaming video and audio. Content may be used to describe a structure, a context or a data item within the context.
Context	Describes the relationships among data items within a structure.
Context Editor	A program in the AltoStudio which specifies the Context of the client. Typically, this involves the linkage of visual items to visual tags representing data items.

**TABLE 4. Glossary (Continued)**

Name	Definition
CORBA	Common Object Request Broker Architecture.
Cube	A user of the server that enables viewing data and relationships in a graphical way by mapping them on to a cube.
Data Connector	A series of programs in AltoServer used connect to specific underlying data sources.
DataInstance	The standard data container for results used by AltoServer. It relates to a MetaObject that describes it. A DataInstance is the result returned by a query.
Data Model	A paradigm which is used to represent data access to a database.
DataObject: DataSet Interface	Encapsulates a group of data entries such as a database table.
DataObject: DataSource Interface	Encapsulates creating a connection to a data source such as Oracle.
DataObject: Dynamic- Query Interface	Encapsulates a query on a data source. It returns the standard data format of AltoServer.
DataObject: Field Interface	Encapsulates a data unit in a DataSet such as a column in a table.
DataObjects	A set of interfaces that represent real data sources (Oracle) and map them to Java objects in AltoServer.
DataSet Interface	A structured series of relationships in a data base such as a table.
DataSource	An interface which encapsulates a data source such as a RDMS, ERP system, multimedia, network, web server data, references and links, file systems and legacy systems.
DataSource Interface	A connection to a specified data source.
DataView Entity/Session Beans	Beans that define the user's visual (or multimedia) experience in the application.
Developer	An advanced user of the system who has permission to create or change the data model that is saved on AltoServer. Works with Alto-Studio, or through a programming API.

TABLE 4. Glossary (Continued)

Name	Definition
DeveloperSession (Bean)	A stateful EJB that is associated with a developer connection from AltoStudio to AltoServer. It gives AltoStudio all the information for building an application, and saves application files to AltoServer.
Dynamic Query	A request for information which is restricted by specifying parameters, ranges, parents and other criteria to limit the possible query responses. One of the two types of query used by AltoServer.
AltoServer Container	The software environment in which EJBs run and which provides system level services.
Enterprise Java Beans (EJBs)	A Java server side services framework from which vendors can create EJB server implementations.
Entity Bean	An EJB bean which is used to represent underlying objects.
Facets	Describes the logical relationships between items in the context window represented by various visual properties.
Field	An element of a relational data base such as a column in a table.
FTOP	File Transfer Operations Protocol.
HTML Servlet	An AltoServer program that connects to an HTML application.
IIOP	Internet Inter-ORB Protocol.
Java Database Connectivity (JDBC)	An application program interface (API) specification for connecting programs written in Java to the data in popular databases (RDBMS).
Java Server Page (JSP)	A technology for controlling the content or appearance of Web pages through the use of servlets, small programs that are specified in the Web page and run on the Web server to modify the Web page before it is sent to the user who requested it.
Layout Entity/Session Bean	Beans which perform the layout (visual/functional interface) of the program.
Layout Session	A program which manages the client-server communication managing the visual and functional parts of the interface (as opposed to data).
MetaCluster	A program in the AltoServer which consists of MetaObjects used to process client applications. The MetaCluster also contains Project and Object Connection data. All application files defined by AltoStudio are stored in the MetaCluster.
Meta Model	A paradigm which expresses the business logic of the application.

**TABLE 4. Glossary (Continued)**

Name	Definition
MetaObject: MetaObject Class	Represents an abstract object. It has properties, logic and connections to other objects.
MetaObject: ObjectConnec- tion Class	Represents the connection between two MetaObjects.
MetaObjects	A set of classes that represents a data map that a developer defines. The instances of this class create a model that is independent of existing data.
MetaProject	An entity bean that saves a project information. Project information consists of MetaObjects, Object Connections and DataObjects.
ObjectConnec- tion	A relationship defined between MetaObjects to define a database connection.
Package	A subdivision of the AltoServer software in a functional unit.
Project	The context in which a complete model is kept. Used by QuerySessions and modified by Developer Sessions. Saved in MetaProject.
Query	A request for authoritative information.
QueryAction- Logic	A Logic Model specifying mathematical or Boolean operations, which is associated with a metaObject which defines logical operations to be performed on data instances or DILists as they are returned from the data base.
QuerySession	The program on the AltoServer session bean which initiates, manages and returns queries.
QueryEvent	A request to initiate a query sent out as defined by a MetaObject or object connection.
Relationship Diagram	A diagram which defines the Object Connections between MetaObjects in the Project.
Rendering	A technology either sphere, cube or HTML used in the client to implement applications.
Remote Method Invo- cation (RMI)	A method, which uses the Java programming language, which allows objects on different computers to interact in a distributed network.
Secure Sockets	A method of communication between client and server in a network created and used with a set of "function calls", which is protected by security features.

TABLE 4. Glossary (Continued)

Name	Definition
Servlet	A small program that is part of a server. Servlets using a Java virtual machine running in the server can execute faster than larger programs such as CGI applications because they are invoked as a thread in a single daemon process, requiring low processor overhead.
Session Bean	An EJB bean where each instance created through its home interface and is private to its client connection.
Site Map	A diagram used in AltoStudio that shows the screens that can be accessed in the application and the relationships between each. Each screen is designated a stage, i.e. Stage 1, Stage 2, Stage 3, etc.
SMTP	Simple Mail Transfer Protocol. The protocol governing network management and monitoring of network devices and their functions in TCP/IP networks.
Sphere	A user of the server that enables viewing data and relationships in a graphical way by mapping them onto a sphere.
Stateless/Stateful Bean	A session bean's declaration descriptor must declare a session bean as either stateless or stateful. A stateless bean is one which does not maintain any state information between method calls. In general, the benefit of a session bean is that it does not maintain state on behalf of the client.
Structure	Encapsulates high level relationships within a Project or MetaObject.
Template	The unchanging visual and functional representation used by a client.
Template Editor	A program in the AltoStudio which specifies the Template (the visual basis) of the client.
Thin Client	An application program that runs on a PC that is designed to be centrally managed, configured and executed by a server program on a network.
Underlying Data Source	A source of data available for client access. Underlying Data Sources may be database, file sources, multimedia sources or ERP data sources, Web data sources, etc. They may have different format and access requirements.
User	A client of AltoServer who connects to it and views the data in a project that was created by the developer. The user has access to a limited part of the server, as defined by the developer.
UserSession	A lightweight stateful session EJB that represents user login to the server.

**TABLE 4. Glossary (Continued)**

Name	Definition
Wrapper	A program or script that makes possible the running of another, more significant program.
XML Servlet	A server program that connects to an XML application or data source.

005090 ET 00209



Yellow X10112



Yellow X10112



# Business Process

